e-f

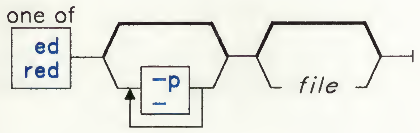
A5AC5024

ed

Purpose

Edits text by line.

Syntax



OL805182

Description

The **ed** command is a line editing program that works on only one file at a time by copying it into a temporary file buffer and making changes to that copy. **ed** does not alter the file itself until you use the write (**w**) subcommand. You can specify on the command line the *file* you want to edit, or you can use the **e**dit subcommands.

When **ed** reads a new file into the buffer, the contents of that file replaces the buffer's previous contents, if any.

There is also a restricted version of **ed**, the **red** command, for use with the restricted shell (see "**sh**" on page 913). With **red**, you can edit only files that reside in the current directory, or in the **/tmp** directory, and you cannot use the !*AIX-cmd* subcommand (see page 383).

An **ed** subcommand consists of zero, one, or two ***addresses***, followed by a single-character subcommand, possibly followed by parameters to that subcommand. These addresses specify one or more lines in the buffer. Because every subcommand has default addresses, you frequently do not need to specify addresses.

The **ed** program operates in one of two modes, ***command mode*** and ***text mode***. In command mode, **ed** recognizes and executes subcommands. In text mode, **ed** adds text to the file buffer but does not recognize subcommands. To leave text mode, enter a **.** (period) alone at the beginning of a line.

Pattern Matching

The **ed** command supports a limited form of ***special pattern-matching characters*** that you can use as ***regular expressions*** (***REs***) to construct ***pattern strings***. You can use these patterns in addresses to specify lines and in some subcommands to specify portions of a line.

Regular Expressions (REs)

The following REs match a *single* character:

char An ordinary *char*acter (one other than one of the special pattern-matching characters), matches itself.

. A . (period) matches any single character except for the new-line character.

[*string*] A *string* enclosed in [ ] (square brackets) matches *any one* character in the string. Certain pattern-matching characters have special meanings within square brackets:

^ If the first character of *string* is a ^ (circumflex), the RE ([^*string*]) matches any character ***except*** the characters in *string* and the new-line character. A ^ has this special meaning ***only*** if it occurs first in the string.

- You can use a - (minus) to indicate a range of consecutive ASCII characters according to the current collating sequence. For example, `[a-f]` might be equivalent to `[abcdef]` or `[aAbBcCdDeEfF]` or `[aáàbcdeéèf]`. The collating sequence is defined by the environment variable **NLCTAB** or **NLFILE**. See *Managing the AIX Operating System* for more information. A collating sequence may define equivalence classes for characters. For example, if three characters—**e**, **é**, and **è**—are equivalent, the following expressions identify the same sequence of characters:

```
[a-e]
[a-è]
```

The minus character loses its special meaning if it occurs first ([-*string*]), if it immediately follows an initial circumflex ([^-*string*]), or if it appears last ([*string*-]) in the string.

] When the right square bracket (]) is the first character in the string ([]*string*]) or when it immediately follows an initial circumflex ([^]*string*]), it is treated as a part of the string rather than as the string terminator.

Japanese Language Support Information

Japanese Language Support introduces a ***character class expression***, in addition to the standard range expression, see 375.

For information about how a string enclosed in square brackets matches characters, see 375.

\sym A \ (backslash) followed by a special pattern-matching character matches the special character itself (as a literal character). These special pattern-matching characters are:

. * [\	Always special *except* when they appear within square brackets ([]).
^	Special at the *beginning* of an entire pattern or when it immediately follows the left bracket of a pair of brackets ([^ . . .]).
$	Special at the *end* of an entire pattern.

In addition, the character used to delimit an entire pattern is special for that pattern. (For example, see how / (slash) is used in the **g** subcommand on page 379.)

Forming Patterns

The following rules describe how to form patterns from REs:

1. An RE that consists of a single, ordinary character matches that same character in a string.

2. An RE followed by an * (asterisk) matches zero or more occurrences of the character that the RE matches. For example, the following pattern:

   ```
   ab*cd
   ```

   matches each of the following strings:

   ```
   acd
   abcd
   abbcd
   abbbcd
   ```

   but not the following string:

   ```
   abd
   ```

If there is any choice, the longest matching leftmost string is chosen. For example, given the following string:

```
122333444
```

the pattern `.*` matches `122333444`, the pattern `.*3` matches `122333`, and the pattern `.*2` matches `122`.

3. An RE followed by:

\\{*m*\\}	Matches *exactly m* occurrences of the character matched by the RE.
\\{*m,*\\}	Matches *at least m* occurrences of the character matched by the RE.
\\{*m,n*\\}	Matches *any number* of occurrences of the character matched by the RE *from m to n inclusive.*

   m and *n* must be integers from 0 to 255, inclusive. Whenever a choice exists, this pattern matches as many occurrences as possible.

4. You can combine REs into patterns that match strings containing that same sequence of characters. For example, `AB\*CD` matches the string `AB*CD` and `[A-Za-z]*[0-9]*` matches any string that contains any combination of alphabetic characters (including none), followed by any combination of numerals (including none).

5. The character sequence \\(*pattern*\\) marks a ***subpattern*** that matches the same string it would match if it were not enclosed.

6. The characters \num match the same string of characters that a subpattern matched earlier in the pattern (see the preceding discussion of item 5). *num* is a digit. The pattern \num matches the string matched by the *num*th subpattern, counting from left to right. For example, the following pattern:

```
\(A\)\(B\)C\2\1
```

   matches the string `ABCBA`. You can nest subpatterns.

Restricting What Patterns Match

A pattern can be restricted to match only the first segment of a line, the final segment, or both:

1. A ^ (circumflex) at the beginning of a pattern causes the pattern to match only a string that begins in the first character position on a line.

2. A $ (dollar sign) at the end of a pattern causes that pattern to match only a string that ends with the last character (not including the new-line character) on a line.

3. The construction ^*pattern*$ restricts the pattern to matching only an entire line.

In addition, the null pattern (that is, //) duplicates the previous pattern.

Japanese Language Support Information

Several characters can have the same collating value; for instance, the ASCII letter a and the SJIS roman letter a could be collated together. In such a case, the expression [a-a] would match both the ASCII and the SJIS roman letters.

You can mix ASCII and SJIS characters in a range expression, as long as the character preceding the minus sign collates equal to or lower than the character following the minus sign. If the character preceding the minus collates higher than the character following the sign, the system interprets the range as consisting only of the two end points.

A common use of the range expression is to match a character class. For example, [0-9] is used to mean all digits, and [a-z A-Z] is used to mean all letters. This form may produce unexpected results when ranges are interpreted according to the current collating sequence.

Instead of the preceding form, use a ***character class expression*** within brackets to match characters. The system interprets this type of expression according to the current character class definition. However, you cannot use character class expressions in range expressions.

Following is the syntax of a character class expression:

[:*charclass*:]

that is, a left bracket, followed by a colon, followed by the name of the character class, followed by another colon and a right bracket.

Japanese Language Support supports the following character classes:

[:upper:]	ASCII uppercase letters
[:lower:]	ASCII lowercase letters
[:alpha:]	ASCII uppercase and lowercase letters
[:digit:]	ASCII digits
[:alnum:]	ASCII alphanumeric characters.
[:xdigit:]	ASCII hexadecimal digits
[:punct:]	ASCII punctuation character (neither a control character nor alphanumeric)
[:space:]	ASCII space, tab, carriage return, new-line, vertical tab, or form-feed character
[:print:]	ASCII printing character

[:jalpha:]	SJIS roman characters
[:jdigit:]	SJIS Arabic digits
[:jxdigit:]	SJIS hexadecimal digits
[:jparen:]	SJIS parentheses characters
[:jpunct:]	SJIS punctuation characters
[:jspace:]	SJIS space characters
[:jprint:]	SJIS printing characters
[:jkanji:]	kanji characters
[:jhira:]	Full-width hiragana characters
[:jkana:]	Half-width and full-width katakana characters

The brackets are part of the character class definition. To match any uppercase ASCII letter or ASCII digit, use the following regular expression:

```
[[:upper:] [:digit:]]
```

Do not use the expression [A-Z0-9].

End of Japanese Language Support Information

Addressing

There are three types of **ed** addresses: line number addresses, addresses relative to the current line, and pattern addresses. The ***current line*** (usually the last line affected by a command) is the point of reference in the buffer. This is the default address for several **ed** commands. (See "Subcommands" on page 378 to find out how each subcommand affects the current line.)

Following are guidelines for constructing addresses:

1. . (dot) addresses the current line.
2. $ (dollar sign) addresses the last line of the buffer.
3. *n* addresses the *n*th line of the buffer.
4. *'x* addresses the line marked with a lowercase ASCII letter, *x*, by the **k** subcommand (see page 380).
5. */pattern/* (a pattern enclosed in slashes) addresses the next line contains a matching string. The search begins with the line after the current line and stops when it finds a match for the pattern. If necessary, the search moves to the end of the buffer, wraps

around to the beginning of the buffer, and continues until it either finds a match or returns to the current line.

6. ?*pattern*? (a pattern enclosed in question marks) addresses the previous line that contains a match for the pattern. The ?*pattern*? construct, like /*pattern*/, can search the entire buffer, but it does so in the opposite direction.
7. An address followed by +*n* or -*n* (a plus sign or a minus sign followed by a decimal number) specifies an address plus or minus the indicated number of lines. (The + sign is optional.)
8. An address that begins with + or - specifies a line relative to the current line. For example, `-5` is the equivalent of `.-5` (five lines above the current line).
9. An address that ends with - or + specifies the line immediately before (-) or immediately after (+) the addressed line. Used alone, the - character addresses the line immediately before the current line. The + character addresses the line immediately after the current line; however, the + character is optional. The + and - characters have a cumulative effect; for example, the address -- addresses the line two lines above the current line.
10. For convenience, a **,** (comma) stands for the address pair `1,$` (first line through last line) and a ; (semicolon) stands for the pair `.,$` (current line through last line).

Commands that do not accept addresses regard the presence of an address as an error. Commands that do accept addresses can use either given or default addresses. When given more addresses than it accepts, a command uses the last (rightmost) one(s).

In most cases, commas (`,`) separate addresses (for example `2,8`). Semicolons (`;`) also can separate addresses. A semicolon between addresses causes **ed** to set the current line to the first address and then calculate the second address (for example, to set the starting line for a search based on rules 5 and 6 above). In a pair of addresses, the first must be numerically smaller than the second.

For many purposes, you may prefer to use a different editor that has different features. Refer to the following discussions for more information:

- "**edit**" on page 387, a simple line editor for novice or casual users
- "**sed**" on page 887, a stream editor often used for writing programs
- "**ex**" on page 407, an extended (line) editor with interactive subcommand features
- "**vi, vedit, view**" on page 1187, a visual (screen) editor that also accesses **ex** line editing features while letting you view the text.

The following is a list of **ed** size limitations:

- 64 characters per file name.
- 512 characters per line (although there is currently a system-imposed limit of 255 characters per line entered from the keyboard).
- 256 characters per global subcommand list.

- 128K characters buffer size. (Note that the buffer not only contains the original file but also editing information. Each line occupies one word in the buffer.)

The maximum number of lines depends on the amount of memory available to you. The maximum file size depends on the amount of physical data storage (disk or tape drive) available or on the maximum number of lines permitted in user memory.

Subcommands

In most cases, only one **ed** subcommand can be entered on a line. The exceptions to this rule are the **p** and **l** subcommands, which can be added to any subcommand except **e**, **f**, **r**, or **w**. The **e**, **f**, **r**, and **w** subcommands accept file names as parameters. The **ed** program stores the last file name used with a subcommand as a default file name. The next **e**, **f**, **r**, or **w** given without a file name uses the default file name.

The **ed** program responds to an error condition with one of two messages: ? (question mark) or ?*file*. When **ed** receives an **INTERRUPT** signal (**Alt-Pause**), it displays a ? and returns to command mode. When **ed** reads a file, it discards ASCII NULL characters and all characters after the last new-line character. **ed** cannot edit a file that contains characters not in the ASCII set (for example, an **a.out** file with bit 8 set on).

In the following list of **ed** subcommands, default addresses are shown in parentheses. Do not key in the parentheses. The address . (period) refers to the current line. When a . is shown in the first position on an otherwise empty line, it is the signal to return to command mode.

(.)**a**
<*text*>
.

The **a**ppend subcommand adds text to the buffer after the addressed line. The **a** subcommand sets the current line to the last inserted line, or, if no lines were inserted, to the addressed line. Address 0 causes the **a** subcommand to add text at the beginning of the buffer.

(.)**c**
<*text*>
.

The **c**hange subcommand deletes the addressed lines, then replaces them with new input. The **c** command sets the current line to the last new line of input, or, if there were none, to the first line that was not deleted.

(.,.)**d**

The **d**elete subcommand removes the addressed lines from the buffer. The line after the last line deleted becomes the current line. If the deleted lines were originally at the end of the buffer, the new last line becomes the current line.

e *file*	The **e**dit subcommand first deletes any contents from the buffer, then loads another file into the buffer, sets the current line to the last line of the buffer, and displays the number of characters read in to the buffer. If the buffer has been changed since its contents were last saved (with the **w** subcommand), **e** displays a ? (question mark) before it clears the buffer. The **e** subcommand stores *file* as the default file name to be used, if necessary, by subsequent **e**, **r**, or **w** subcommands. (See the **f** subcommand.) When an ! (explanation mark) replaces *file*, **e** takes the rest of the line as a AIX shell (**sh**) command and reads the command output. The **e** subcommand does not store the name of the shell command as a default file name.
E *file*	The **E**dit subcommand works like **e**, with one exception: **E** does not check for changes made to the buffer since the last **w** subcommand.
f [*file*]	The **f**ile name subcommand changes the default file name (the stored name of the last file used) to *file*, if *file* is given. If *file* is not given, the **f** subcommand prints the default file name.
(1,?)**g**/*pattern*/*subcmd-list*	The **g**lobal subcommand first marks every line that matches the pattern. Then, for each marked line, this subcommand sets the current line to that line and executes *subcmd-list*. A single subcommand, or the first subcommand of a list, should appear on the same line with the **g** subcommand; subsequent subcommands should appear on separate lines. Except for the last line, each of these lines should end with a \\. The *subcmd-list* can include the **a**, **i**, and **c** subcommands and their input. If the last command in *subcmd-list* would normally be the . (period) that ends input mode, the . is optional. If there is no *subcmd-list*, **ed** displays the current line. The *subcmd-list* cannot include the **g**, **G**, **v**, or **V** subcommands. **Note:** The **g** subcommand is similar to the **v** subcommand, which executes *subcmd-list* for every line that does not contain a match for the pattern.
(1,?)**G**/*pattern*/	The interactive **G**lobal subcommand first marks every line that matches the pattern, then displays the first marked line, sets the current line to that line, and waits for a subcommand. **G** accepts any but the following **ed** subcommands: **a**, **c**, **i**, **g**, **G**, **v**, and **V**. After the subcommand finishes, **G** displays the next marked line, and so on. **G** takes a new-line character as a null subcommand. A :& (colon ampersand) causes **G** to execute the

	previous subcommand again, if there was one. Note that subcommands executed within the **G** subcommand can address and change any lines in the buffer. The **G** subcommand can be terminated by pressing **INTERRUPT** (**Alt-Pause**).
h	The **h**elp subcommand gives a short explanation (help message) for the most recent ? diagnostic or error message.
H	The **H**elp subcommand causes **ed** to display the help messages for all subsequent ? diagnostics. **H** also explains the previous ? if there was one. **H** alternately turns this mode on and off; it is initially off.
(.)**i** < *text* > **.**	The **i**nsert subcommand inserts text before the addressed line and sets the current line to the last inserted line. If there no lines are inserted, **i** sets the current line to the addressed line. This subcommand differs from the **a** subcommand only in the placement of the input text. Address **0** is not legal for this subcommand.
(.,.+1)**j**	The **j**oin subcommand joins contiguous lines by removing the intervening new-line characters. If given only one address, **j** does nothing. (For splitting lines, see the **s** subcommand.)
(.)**k***x*	The mar**k** subcommand marks the addressed line with name *x*, which must be a lowercase ASCII letter. The address *'x* (single quotation mark before the marking character) then addresses this line. The **k** subcommand does not change the current line.
(.,.)**l**	The **l**ist subcommand displays the addressed line(s). The **l** subcommand wraps long lines and, unlike the **p** subcommand, represents non-printing characters, either with mnemonic overstrikes or in hexadecimal notation. An **l** subcommand may be appended to any **ed** subcommand except: **e**, **f**, **r**, or **w**.
(.,.)**m***a*	The **m**ove subcommand repositions the addressed line(s). The first moved line follows the line addressed by *a*. Address 0 for *a* causes **m** to move the addressed line(s) to the beginning of the file. Address *a* cannot be one of the lines to be moved. The **m** subcommand sets the current line to the last moved line.
(.,.)**n**	The **n**umber subcommand displays the addressed lines, each preceded by its line number and a tab character (displayed as blank spaces); **n** leaves the current line at the last line displayed. An **n** subcommand may be appended to any **ed** subcommand except **e**, **f**, **r**, or **w**.

(.,.)**p**	The **p**rint subcommand displays the addressed line(s) and sets the current line set to the last line displayed. A **p** subcommand may be appended to any **ed** subcommand except: **e**, **f**, **r**, or **w**. For example, the subcommand **dp** deletes the current line and displays the new current line.
P	The **P** subcommand turns on or off the **ed** prompt string * (asterisk). Initially, **P** is off.
q	The **q**uit subcommand exits the **ed** program. Before ending the program **q** checks to determine whether the buffer has been written to a file since the last time it was changed. If not, **q** displays the **?** message.
Q	The **Q**uit subcommand exits the **ed** program without checking for changes to the buffer since the last **w** subcommand (compare with the **q** subcommand).
(?)**r** *file*	The **r**ead subcommand reads a file into the buffer after the addressed line; **r** does not delete the previous contents of the buffer. When entered without *file*, **r** reads the default file, if any, into the buffer (see **e** and **f** subcommands). **r** does not change the default file name. Address 0 causes *r* to read a file in at the beginning of the buffer. After it reads a file successfully, **r**, displays the number of characters read into the buffer and sets the current line to the last line read. If ! (exclamation point) replaces *file* in a **r** subcommand, **r** takes the rest of the line as a AIX shell (**sh**) command whose output is to be read. The **r** subcommand does not store the names of shell commands as default file names.
(.,.)**s**/*pattern*/*replacement*/ (.,.)**s**/*pattern*/*replacement*/**g**	The **s**ubstitute subcommand searches each addressed line for a string that matches the pattern and then replaces the string with the specified *replacement* string. Without the global indicator (**g**), **s** replaces only the first matching string on each addressed line. With the **g** indicator, **s** replaces every occurrence of the matching string on each addressed line. If **s** does not find a match for the pattern, it returns the error message ?. Any character except a space or a new-line character can separate (delimit) the pattern and *replacement*. The **s** subcommand sets the current line to the last line changed. An & (ampersand) in the *replacement* string is a special symbol that has the same value as the *pattern* string. For example, the subcommand **s/are/&n't/** has the same effect as the subcommand **s/are/aren't/** and replaces **are** with **aren't** on

the current line. A \& (backslash ampersand) removes this special meaning of & in *replacement.*

A subpattern is part of a pattern enclosed by the strings \(and \); the pattern works as if the enclosing characters were not present. In *replacement*, the characters *n* refer to strings that match subpatterns; *n*, a decimal number, refers to the *n*th subpattern, counting from the left. (for example, **s/\(t\)\(h\)\(e\)/t\1\2ose)** replaces **the** with **those** if there is a match for the pattern **the** on the current line). Whether subpatterns are nested or in a series, *n* refers to the *n*th occurrence, counting from the left, of the delimiting characters, \).

The % (percent sign) character, when used by itself as *replacement*, causes **s** to use the previous *replacement* again. The % character does not have this special meaning if it is part of a longer *replacement* or if it is preceded by a \.

Lines may be split by substituting new-line characters into them. In *replacement*, the sequence **Enter** quotes the new-line character (not displayed) and moves the cursor to the next line for the remainder of the string. New-lines cannot be substituted as part of a **g** or **v** subcommand list.

(.,.)**t***a*
The **t**ransfer subcommand inserts a copy of the addressed lines after address *a*. The **t** subcommand accepts address 0 (for inserting lines at the beginning of the buffer). The **t** subcommand sets the current line to the last line copied.

u
The **u**ndo subcommand restores the buffer to the state it was in before it was last modified by an **ed** subcommand. The commands that **u** can undo are: **a**, **c**, **d**, **g**, **G**, **i**, **j**, **m**, **r**, **s**, **t**, **v**, and **V**.

(1,?)**v**/*pattern*/*subcmd-list*
The **v** subcommand executes the subcommands in *subcmd-list* for each line that does not contain a match for the pattern.

Note: The **v** subcommand is a complement for the global subcommand **g**, which executes *subcmd-list* for every line that does contain a match for the pattern.

(1,$)**V**/*pattern*//
The **V** subcommand first marks every line that does not match the pattern, then displays the first marked line, sets the current line to that line, and waits for a subcommand.

Note: The **V** subcommand complements the **G** subcommand, which marks the lines that do match the pattern.

(1,?)**w** *file*	The **write** subcommand copies the addressed lines from the buffer to the file named in *file*. If the file does not exist, the **w** subcommand creates it with permission code 666 (read and write permission for everyone), unless the **umask** setting specifies another file creation mode. (For information about file permissions, see "**umask**" on page 1110 and "**chmod**" on page 160.) The **w** subcommand does not change the default file name (unless *file* is the first file name used since you started **ed**). If you do not provide a file name, **ed** uses the default file name, if any (see the **e** and **f** subcommands). The **w** subcommand does not change the current line. If **ed** successfully writes the file, it displays the number of characters written. When ! replaces *file*, **ed** takes the rest of the line as a AIX shell (**sh**) command whose output is to be read; **w** does not save shell command names as default file names. **Note:** 0 is not a legal address for the **w** subcommand. Therefore, it is not possible to create an empty file with **ed**.
($)=	Without an address, the = (equal sign) subcommand displays the current line number. With the address $, = displays the number of the last line in the buffer. The = subcommand does not change the current line and cannot be included in a **g** or **v** subcommand list.
!*AIX-cmd*	The ! (exclamation point) subcommand allows AIX commands to be run from within **ed**. Anything following **!** on an **ed** subcommand line is interpreted as an AIX command. Within the text of that command string, **ed** replaces the unescaped % (percent sign) with the current file name, if there is one. When used as the first character of a shell command (after the ! that runs a subshell) **ed** replaces the ! character with the previous AIX command; for example, the command **!!** repeats the previous AIX command. If the AIX command interpreter (the **sh** command), expands the command string, **ed** echoes the expanded line. The ! subcommand does not change the current line.
num +*num* -*num*	**ed** interprets a number alone on a line as an address and displays the addressed line. Addresses can be absolute (line numbers or $) or relative to the current line (+*num* or - *num*). Entering a new-line character (a blank line) is equivalent to

+**1p** and is useful for stepping forward through the buffer one line at a time.

Flags

-	Suppresses character counts that the editor displays with the **e**, **r**, and **w** subcommands, suppresses diagnostic messages for the **e** and **q** subcommands, and suppresses the ! prompt after a **!***AIX-cmd*.
-p *string*	Sets the editor prompt to *string*. The default for *string* is null (no prompt).

Files

/tmp/e#	Temporary file; # is the process number.
ed.hup	Work is saved here if the terminal hangs up while **ed** is running.

Related Information

The following commands: "**grep**" on page 501, "**sed**" on page 887, "**sh**" on page 913, "**stty**" on page 1018, and "**regcmp**" on page 820.

The **regexp** system call in *AIX Operating System Technical Reference.*

The **environment** miscellaneous facility in *Text Formatting Guide.*

The discussion and examples of **ed** in *Using the AIX Operating System.*

"Overview of International Character Support" in *Managing the AIX Operating System.*

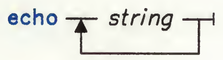
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

echo

Purpose

Writes its arguments to standard output.

Syntax



OL805115

Description

The **echo** command writes its arguments to standard output. *string*s are separated by blanks and a new-line character follows the last *string*. Use **echo** to produce diagnostic messages in command files and to send data into a pipe.

The **echo** command recognizes the following escape conventions:

\b	Display a backspace character.
\c	Suppress the new-line character.
\f	Display a form-feed character.
\n	Display a new-line character.
\r	Display a carriage return character.
\t	Display a tab character.
\\	Display a backslash character.
\num	Display an 8-bit character whose ASCII value is the 1-, 2- or 3-digit octal number *num*. The first digit of *num* must be a zero.

Examples

1. To write a message to standard output:

```
echo Please insert diskette . . .
```

2. To display a message containing special characters:

   ```
   echo "\n\n\nI'm at lunch.\nI'll be back at 1:00."
   ```

   This skips three lines and displays the message:

   ```
   I'm at lunch.
   I'll be back at 1:00.
   ```

   Note: You must put the message in quotation marks if it contains escape sequences like `\n`. Otherwise, the shell treats the \ specially. See page 918 for details about quoting.

3. To use **echo** with pattern-matching characters:

   ```
   echo The back-up files are: *.bak
   ```

   This displays the message `The back-up files are:` followed by the file names in the current directory ending with `.bak`.

4. To add a single line of text to a file:

   ```
   echo Remember to set the shell search path to $PATH. >>notes
   ```

   This adds the message to the end of the file `notes` after the shell substitutes the value of the shell variable **PATH**.

5. To write a message to the standard error output:

   ```
   echo Error: file already exists. >&2
   ```

   Use this in shell procedures to write error messages. If the `>&2` is omitted, then the message is written to the standard output. For details about this type of file redirection, see "Input and Output Redirection Using File Descriptors" on page 928.

Related Information

The following commands: "**csh**" on page 225 and "**sh**" on page 913.

Note: The **csh** command contains a built-in subcommand named **echo**. The command and subcommand do not necessarily work the same way. For information on the subcommand, see the **csh** command.

edconfig

Purpose

Edits values in a **sendmail** configuration file.

Syntax

```
/usr/lib/edconfig —— file ——|
```

AJ2FL249

Description

The **edconfig** command allows you to edit a specified configuration file for the **sendmail** program. It provides a menu interface to changing some of the characteristics defined in the configuration file. To change other characteristics, you must use a text editor. You must have *superuser* authority to edit the configuration file that the system uses (**/usr/adm/sendmail/sendmail.cf**). The **edconfig** command allows you to define or change the following types of entries in the configuration file:

- The content of the *host name* class and macro
- The *domain name* macro
- The four classes that define the separate tokens of the domain name
- Configuration options (with help information) for:
  - Operational logging level
  - Default delivery mode
  - Alias file path
  - Statistics file path
  - Queue file path
  - Maximum mail retention time in queue
  - Queue uses of expensive mailers
- Configuration File Revision level

The *file* parameter provides the path name of the configuration file that you want to edit. The file must be in the format of the configuration file that is supplied with the operating system. The program searches for comment lines in that file of the form *#parameter* to locate the information in that file concerning *parameter*. For example, the comment line `#Or` precedes the line that defines the read timeout option.

To change parameters in the standard **sendmail** configuration file, enter the following command (while operating with superuser authority):

```
edconfig /usr/adm/sendmail/sendmail.cf
```

The program reads the contents of the configuration file into memory. It then displays a menu to help you select what to change. All changes are made only to the copy of the file in memory until you choose to exit and write the changes to the file. You can also exit without writing changes. The program provides information with each step in the menus to help you decide how to enter the correct information. However, you should be familiar with the **sendmail** program and its use before changing information in the configuration file.

Files

/usr/lib/edconfig	The **edconfig** program.
/usr/lib/edconfig.hf	A text file containing the help information that **edconfig** displays.
/usr/adm/sendmail/sendmail.cf	The configuration file for the **sendmail** program.

Related Information

"**sendmail**" on page 897.

The chapter about managing the mail system in *IBM RT Managing the AIX Operating System.*

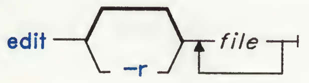
The entry for **sendmail.cf** in *AIX Operating System Technical Reference.*

edit

Purpose

Provides a simple line editor for the new user.

Syntax



OL805329

Description

The **edit** command provides a line editor designed for beginning users. It is a simplified version of the **ex** command (see "**ex**" on page 407). To edit the contents of a file, enter:

`edit` *file*

If *file* is the name of an existing file, **edit** copies it to a buffer and displays the number of lines and characters in it. Then it displays a **:** (colon) prompt to show that it is ready to read subcommands from standard input. If *file* does not already exist, **edit** tells you this, but still stores the name as the current file name. You can give more than one *file* name, in which case **edit** copies the first file into its buffer and stores the remaining file names in an ***argument list*** for later use.

The **edit** command operates in one of two modes: ***command mode*** and ***text entry mode***. In command mode, **edit** displays the colon prompt to show you that it is ready to accept **edit** subcommands. In text entry mode, **edit** places all input into its editing buffer. The general format of an **edit** subcommand is as follows:

[*addr*]*subcommand* [*parameters*] [*count*]

If you do not specify an *addr*ess, **edit** works on the ***current line***. If you add a numeric *count* to most subcommands, **edit** works on the specified number of lines.

For most subcommands, the last line affected becomes the new current line. That means, for example, that after **edit** reads a file into its buffer, the last line in the file becomes the current line. *addr* can be a line number or a *pattern* to be matched or, in some cases, a range of line numbers or *pattern*s. To specify a range, separate two line numbers or *pattern*s with a comma or a semicolon (for example, `1,5` or `1;5`). In a range, the second address must refer to a line that follows the first addressed line in the range.

Addressing Lines within a File

The simplest way to address a line within a file is to use its line number. But this can be unreliable because line numbers change when you insert and delete lines. **edit** provides a way to search through the buffer for strings. Given the following address:

/pattern/

edit searches forward for *pattern*, while given:

?*pattern*?

it searches backwards for *pattern*. If a forward search reaches the end of the buffer without finding *pattern*, it continues the search at the beginning of the file until it reaches the current line. A backwards search does just the reverse.

The following characters have special meanings in these search patterns:

^	Matches the beginning of a line.
$	Matches the end of a line.

Thus, you can use */^pattern/* to search for patterns at the beginning of a line, and */pattern$/* to search for patterns at the end of the line.

The current line has a symbolic name, **.** (period) and the last line in the buffer has a symbolic name, dollar sign ($), that you can use in addresses. This is useful when working with a range of lines. For example,

```
.,$print
```

displays all lines from the current line to the last line in the buffer. Arithmetic with line references is also possible, so that `$-5` refers to the fifth line from the last and `.+20` refers to the line 20 lines past the current line. You can also use the = (equal) command to find out the line number of the current line or the last line, as follows:

```
.=
$=
```

To view the next line in the buffer, press the **Enter** key. Press **Ctrl-D** to display the next half-screen of lines.

Note: Do not confuse the meaning of $ in text patterns (end of line) with its meaning in addresses (last line).

Using a Family of Editors

The **edit** command is part of a family of editors that includes **edit**, **ex**, and **vi**. The **edit** command is a simple line editor designed for beginning users. It is a simplified version of **ex**. After you become more experienced with **edit**, you may want to try the advanced features of one of the other editors in the family. Because **edit** is part of a family of editors, you can apply your knowledge of **edit** to the other editors in the family.

The **ex** editor is a powerful interactive line editor. The **edit** subcommands work the same way in **ex**, but the editing environment is somewhat different. For example in **edit**, only the characters ^, $, and \ have special meanings as pattern-matching characters; however, several additional characters also have special meanings in **ex**. For more information on **ex**, see "**ex**" on page 407.

The **vi** editor is a display-based editor designed for experienced users who edit intensively at their display. It contains many of the advanced features of **ex**, but focuses on the display editing portion of **ex**. The **edit** editor prevents you from accidentally entering **vi**'s two alternative modes of editing, the open mode and the visual mode. For more information on **vi**, see "**vi**, **vedit**, **view**" on page 1187.

Flag

-r Recovers *file* after an editor or system crash.

Subcommands

You can enter most **edit** subcommands as either a complete word or an abbreviation. In the following list, a subcommand abbreviation appears in parentheses. Unless noted otherwise, all subcommands work by default on the current line. **edit** recognizes and interprets the following subcommands when it displays the colon prompt:

[*addr*]**a**ppend
text
.
Reads the input *text* into the file being edited, placing the text after the line at the specified *addr*ess. If you specify address 0, **edit** places the text at the beginning of the buffer. To return to command mode, enter a line with only a . (period) in the first position.

[*addr1*[,*addr2*]]**c**hange
text
.
Replaces the specified line or lines with the input *text*. If any lines are input, the last input line becomes the new current line.

[*addr1*[,*addr2*]]**d**elete [*buffer*]
Removes the specified line or lines from the editing buffer. The line following the last deleted line becomes the current line. If you specify a *buffer* by giving a letter from a to z, **edit** saves the specified lines in that buffer or, if the letter is uppercase, appends the lines to that buffer.

edit *file* Begins an editing session on a new file. The editor first checks to see if the buffer has been modified (***edited***) since the last **write** subcommand. If it has, **edit** issues a warning and cancels the **edit** subcommand. Otherwise, it deletes the complete contents of the editor buffer, makes the named file the current file, and displays the new file name. After insuring that this file can be edited, it reads the file into its buffer. If **edit** reads the file without error, it displays the number of lines and characters that it read. The last line read becomes the new current line.

file
: Displays the current file name along with the following information about it:

- Whether it has been modified since the last **write**.
- What the current line is.
- How many lines are in the buffer.
- What percentage of the way through the buffer the current line is.

file *file*
: Changes the name of the current file to *file*. **edit** considers this file ***not edited***.

[*addr1*[,*addr2*]]global/*pattern*/*cmds*
: Marks each of the specified lines that matches the *pattern*. Then **edit** carries out the specified subcommands (*cmds*) on each marked line.

A single *cmd* or the first *cmd* in a subcommand list appears on same line as **global**. The remaining *cmd*s must appear on separate lines, where each line (except the last) ends with a \ (backslash). The default subcommand is **print**.

The list can include the **append**, **insert**, and **change** subcommands and their associated input. In this case, if the ending period comes on the last line of the command list, you can omit it. The **undo** subcommand and the **global** subcommand itself, however, may not appear in the command list.

[*addr*]insert (i)
text
: Places the given text before the specified line. The last line input becomes the current line. Otherwise, the current line does not change.

[*addr1*[,*addr2*]]move *addr3* (m)
: Repositions the specified line or lines to follow *addr3*. The first of the moved lines becomes the current line.

next (n)
: Copies the next file in the command line argument list to the buffer for editing.

[*addr1*[,*addr2*]]number (nu)
: Displays each specified line or lines preceded by its buffer line number. The last line displayed becomes the current line.

preserve
: Saves the current editor buffer as though the system had just crashed. Use this command when a **write** subcommand has resulted in an error, and you do not know how to save your work.

[*addr1*[,*addr2*]]print (p)
: Displays the specified line or lines. The last line becomes the current line.

[*addr*]put *buffer* (pu)
: Retrieves the contents of the specified buffer and places it after *addr*. If you do not specify a buffer, **edit** restores the last deleted or yanked text. Thus you can use this subcommand together with **delete** to move lines or with **yank** to duplicate lines between files.

quit (q)
quit! (q!) Ends the editing session.

Note: The **quit** command does ***not*** write the editor buffer to a file. However, if you have modified the contents of the buffer since the last **write**, **edit** displays a warning message and does not end the session. In this case, either use the **quit!** subcommand to discard the buffer or **write** the buffer and then **quit**.

recover *file* Recovers *file* from the system save area. Use this after a system crash, or a **preserve** subcommand.

[*addr1*[,*addr2*]]substitute/*pattern*/*repl*/ (s)
[*addr1*[,*addr2*]]substitute/*pattern*/*repl*/g
: Replaces on each specified line the ***first*** instance of *pattern* with the replacement pattern *repl*. If you add the **g** flag, it replaces *all* instances of *pattern* on each specified line.

undo (u) Reverses the changes made in the buffer by the last buffer editing subcommand. Note that **global** subcommands are considered a single subcommand to an **undo**. You cannot **undo** a **write** or an **edit**.

[*addr1*,[*addr2*]]write *file* (w)
: Writes the contents of the specified line or lines to *file*. The default range is all lines in the buffer. **edit** displays the number of lines and characters that it writes. If you do not specify a *file*, **edit** uses the current file name. If *file* does not exist, **edit** creates it.

[*addr1*,[*addr2*]]yank [*buffer*] (ya)
: Places the specified line or lines in *buffer* (a single alpha character name **a** - **z**).

[*addr*]z Displays a screen of text, beginning with the specified line.

[*addr*]z- Displays a screen of text, with the specified line at the bottom of the screen.

[*addr*]z. Displays a screen of text, with the specified line in the middle of the screen.

Related Information

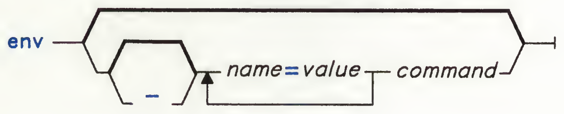
The following commands: "**ed**" on page 371, "**ex**" on page 407, and "**vi, vedit, view**" on page 1187.

env

Purpose

Sets the environment for execution of a command.

Syntax



OL805117

Description

The **env** command lets you get and change your current environment, and then run the named *command* with the changed environment. Changes in the form *name=value* are added to the current environment before the command is run. If - (minus) is used, the current environment is ignored and the command runs with only the changed environment. Changes are only in effect while the named *command* is running.

If a *command* is not specified, **env** displays your current environment one *name=value* pair per line.

Examples

1. To add a shell variable to the environment for the duration of one command:

   ```
   TZ=MST7MDT date
   env TZ=MST7MDT date
   ```

   Each of these commands displays the current date and time in Mountain Standard Time. The two commands shown are equivalent. When **date** is finished, the previous value of **TZ** takes effect again.

2. To replace the environment with another one:

   ```
   env - PATH=$PATH IDIR=/u/jim/include LIBDIR=/u/jim/lib make
   ```

   This runs **make** in an environment that consists *only* of these definitions for PATH, IDIR, and LIBDIR. You must redefine **PATH** so that the shell can find the **make** command.

   When **make** is finished, the previous environment takes effect again.

Related Information

The following command: "**sh**" on page 913.

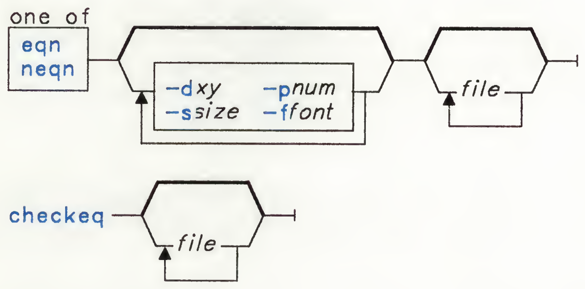
The **exec** system call, the **profile** file, and the **environ** miscellaneous facility in *AIX Operating System Technical Reference.*

eqn, neqn, checkeq

Purpose

Formats mathematical text for the **nroff, troff** and **troff** commands.

Syntax



OL805183

Description

The **eqn** command is a **troff** preprocessor for typesetting mathematical text on a phototypesetter. The **neqn** command is used with **nroff** for other *printing devices*. The output of **eqn** and **neqn** is generally piped into **troff** and **nroff** as follows:

```
eqn file ¦ troff
neqn file ¦ nroff
```

If you do not specify any files or if you specify - as the last file name, the commands read standard input. A line consisting of **.EQ** marks the start of equation text; the end of equation text is marked by a line consisting of **.EN**. Neither of these lines is altered by the commands, so they can be defined in macro packages to give you centering and numbering. The program **checkeq** reports missing or unbalanced delimiter pairs and **.EQ/.EN** pairs. For information on how to format **eqn** text, see *Text Formatting Guide*.

The **eqn** command recognizes the following mathematical words, and prints the associated symbol:

above	**dotdot**	**italic**	**rcol**	**to**
back	**down**	**lcol**	**right**	**under**
bar	**dyad**	**left**	**roman**	**up**
bold	**fat**	**lineup**	**rpile**	**vec**
ccol	**font**	**lpile**	**rpile**	~
col	**from**	**mark**	**size**	^
cpile	**fwd**	**matrix**	**sub**	{}
define	**gfont**	**ndefine**	**sup**	" . . . "
delim	**gsize**	**over**	**tdefine**	
dot	**hat**	**pile**	**tilde**	

Flags

-d*xy* Sets *x* and *y* as one character delimiters of the text to be processed by **eqn**, in addition to the **.EQ** and **.EN** macros. The text between these delimiters will be treated as input to **eqn**.

Note: Within a file, you can also set delimiters for **eqn** text using the command **delim** *xy*. They are turned off by the command **delim off**. All text that is not between delimiters or **.EQ** and **.EN** is passed through unprocessed.

-f*font* Acts the same as **-s** for fonts. See the discussion of **gfont** and **font** in *Text Formatting Guide* for information on changing font within the text.

-p*num* Reduces subscripts and superscripts *num* points in size (the default is 3).

-s*size* Changes point size in all **eqn** processed text to *size*. See the discussion of **gsize** and **size** in *Text Formatting Guide* for information on changing the point size within the text.

Related Information

The following commands: "**cw, checkcw**" on page 275, "**mm, checkmm**" on page 663, "**mmt, checkmm**" on page 666, "**nroff, troff**" on page 709, and "**troff**" on page 710.

The **eqnchar** and **mv** miscellaneous facilities in *AIX Operating System Technical Reference*.

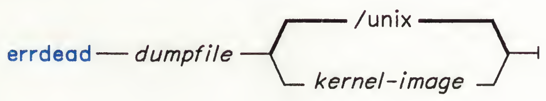
The discussion of **eqn** in *Text Formatting Guide*.

errdead

Purpose

Extracts error records from dump.

Syntax



OL805120

Description

When the system detects a hardware error, it produces an error record containing information pertinent to the error. If **errdemon**, the error-logging daemon, is not running or if the system crashes before it can place the record in the error file, the system holds the error information in a local buffer. **errdead** examines a system dump (or memory), extracts the error records, and passes them to **errpt** to generate a report. Note that no analysis is available because these error entries were never sent back via **errdemon**.

The *dumpfile* parameter specifies the file (or memory) to be examined. The *kernel-image* parameter specifies the system name list, by default **/unix**.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Files

/unix	System kernel image.
/usr/bin/errpt	Analysis program.
/usr/tmp/err*	Temporary file.

Related Information

The following command: "**errpt**, **errpd**" on page 400.

The discussion of **errdead** in *AIX Operating System Programming Tools and Interfaces.*

errdemon

Purpose

Starts the error-logging daemon.

Syntax

```
/usr/lib/errdemon  —┤ 1
```

1 This command is not usually run from the command line.

OL805118

Description

The error-logging daemon **errdemon** collects error records from the operating system by reading the special file **/dev/error** and places them in one of two error log files. **errdemon** creates the names of the two log files by adding a **.0** and **.1** to the end of the file name found in **/etc/rasconf**. If an error log file does not already exist, **errdemon** creates one.

The **errdemon** command adds error records to the first error log file until it reaches the maximum allowable length specified in **/etc/rasconf**. At that point, **errdemon** closes the first error log file, changes the file name from *filename*.**0** to *filename*.**1**, and opens a new *filename*.**0**. Thus, the newest error records are always in *filename*.**0**. When it is full, **errdemon** overwrites the first file.

You can stop the error-logging daemon by sending it a **SIGKILL** signal (see "**errstop**" on page 404). Normally, the **/etc/rc** command file runs **errdemon** at system startup. Only a user operating with superuser authority can start **errdemon**, and only one daemon may be active at any time.

If **errdemon** is unable to log an error, it logs it in abbreviated form in **/dev/nvram**. Just one error can be logged in **/dev/nvram**, so each subsequent error overwrites any previous entries. When the system is started, **errdemon** searches for a previously written entry in **/dev/nvram** and, if a record is found, records it in one of the error log files and clears **/dev/nvram**.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Files

/dev/error	Source of error records.
/dev/nvram	Non-volatile read-only memory.
/etc/rasconf	Configuration file.
/etc/rc	System startup file.
/usr/adm/ras/errfile*	Repository for error records.

Related Information

The following commands: “**errpt**, **errpd**” on page 400, “**errstop**” on page 404, and “**kill**” on page 552.

The **error** and **nvram** files in *AIX Operating System Technical Reference.*

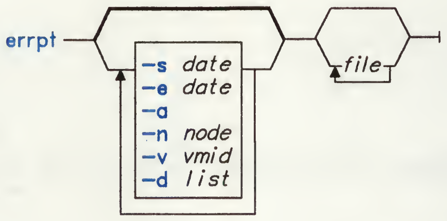
AIX Operating System Programming Tools and Interfaces.

errpt, errpd

Purpose

Processes a report of logged errors.

Syntax



OL805410

Description

The **errpt** command reads a specified error *file* or *file*s, processes the data, and writes a report of that data to standard output. These error files should be named *file*.**0** or *file*.**1**, but do not include the **.0** or **.1** extension when you specify the file name argument. **errpt** adds the extension. If you do not specify a file name, **errpt** uses the file listed in **/etc/rasconf**, adding the **.0** and **.1** extensions (these are usually **/usr/adm/ras/errfile.0** and **/usr/adm/ras/errfile.1**). The default report is a summary of all errors posted in the named file, as well as system information events, such as time changes, system starts, and so on.

The **errpt** command pipes error entries through the program **/usr/lib/errpd**, which adds probable cause information to certain entries. If no probable cause information is added, **errpt** logs records exactly as it receives them.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-a	Produces a detailed report. This contains specific error information for every event that **errpt** formats.

-d *list*	Limits the report to certain types of error records as defined by *list.* The list items can either be separated by commas or enclosed in double quotation marks and separated by commas or blanks. See "Error Identifiers" for the valid list values.
-e *date*	Includes all records posted earlier than *date*, where *date* has the form *MMddhhmmyy* (month, day, hour, minute and year).
-n *node*	Includes only entries in the error report from the specified *nodename.*
-s *date*	Includes all records posted later than *date*, where *date* has the form *MMddhhmmyy.*
-v *vmid*	Includes only entries in the error report from the system name specified with *vmid.*

Error Identifiers

In the following error identifiers, **0** acts as a wildcard character, such that, for example, **H00** gives you all hardware errors (**H11** to **HFF**), and **H10** gives you all errors from **H11** to **H1F**, and so on.

1. Class

   H00 = Hardware (01)
   S00 = Software (02)
   I00 = IPL/Shutdown (03)
   G00 = General System Condition (04)
   U00 = User Defined, Non-Hardware

2. Class/Subclass

   H10 = Hardware/Processor and Memory Management Card Machine Check
   H11 = Hardware/Main Processor
   H12 = Hardware/Main Memory
   H20 = Hardware/Fixed Disk Drive and Adapter
   H30 = Hardware/Diskette Drive and Adapter
   H40 = Hardware/Tape and Adapter
   H50 = Hardware/Display Station
   H51 = Hardware/5080 Display Adapter
   H52 = Hardware/APA16 Display Adapter
   H60 = Hardware/Display Station Adapter
   H70 = Hardware/Keyboard/Mouse
   H80 = Hardware/Communication Adapters
   H81 = Hardware/RS232 Multi-port
   H84 = Hardware/Serial or Serial/Parallel
   H85 = Hardware/IBM PC Network Adapter
   H86 = Hardware/RS422 Multi-port
   H87 = Hardware/Native Serial I/O

H8E = Hardware/SSLA
H90 = Hardware/Parallel Printer and Adapter
H91 = Hardware/Parallel or Serial/Parallel
H92 = Hardware/Parallel or PC Monochrome
HA0 = Hardware/Printers
HF0
.
.
.
HFF = User Defined Hardware

S10 = Software/Processor and Memory Management Card Program Check
S20 = Software/Abend
S21 = Software/Abend dump taken
S22 = Software/Abend No dump taken
S30 = Software/Program Error AIX
S33 = Software/Program Error Kernel
S40 = Software/Program Error Kernel Device Driver
S42 = Software/5080 Display Device Driver
S50 = Software/Program Error Kernel Device Driver
S60 = Software/Program Error VRM Base
S61 = Software/Program Error VRM Attach Device
S70 = Software/Program Base VRM Component
S72 = Software/Program Base VRM Component - Virtual Terminal
S74 = Software/5080 Display VRM Device Driver
S75 = Software/5080 Peripherals VRM Device Driver Manager
S80 = Software/Program Error Application
S80 = Software/Program Error Application - Error Log Analysis
S80 = Software/Program Error Application - Interactive Workstation
S90 = Software/Program Error Application
SA0 = Software/Program Error Application
SB0 = Software/Program Error Application
SC0 = Software/Program Error Application
SD0 = Software/Program Error Application
SE0 = Software/Program Error Application
SF0 = Software/Program Error Application

I10 = IPL/Shutdown/Manual IPL
I20 = IPL/Shutdown/Soft IPL
I30 = IPL/Shutdown/Auto IPL
I40 = IPL/Shutdown/Shutdown
I50 = IPL/Shutdown/Maintenance Shutdown

G10 = General System Condition/Degraded Config
G20 = General System Condition/Set Date/ Time
G40 = General System Condition/Error Reporting
G50 = General System Condition/LPOST

G41	=	General System Condition/Cause Codes
G42	=	General System Condition/Device Information
G43	=	General System Condition/Counters
G51	=	General System Condition/Memory Test LPOST
U10		
.		
.		
.		
UFF	=	User Defined, Non-Hardware

errpd

The error log analysis program, **/usr/lib/errpd**, analyzes the error log data. **/usr/lib/errpd** processes error data to determine if the error is a hardware error and if the error is a temporary or permanent error.

The analysis does the following:

- Generates a number that corresponds to a service request number.
- Analyzes the data and generates the ALERT number.
- Makes the description message ID number. The description consists of the following:
  - Error Analysis determines, from the error data passed, the nature of the operation at the time of the failure. This becomes part of the error description.
  - Error Analysis determines what failed and what the error indication is. This becomes part of the error description and is used to create the ALERT number.
  - Field Replacement Unit (FRU) Analysis determines the Service Request Code. This becomes part of the error description.

Files

/usr/adm/ras/errfile?	Error file.

Related Information

The following command: "**errdemon**" on page 398.

The **errfile** file in *AIX Operating System Technical Reference.*

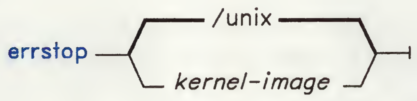
AIX Operating System Programming Tools and Interfaces.

errstop

Purpose

Terminates the error-logging daemon.

Syntax



OL805121

Description

The **errstop** command stops the error-logging daemon **errdemon** by running the **ps** command to determine the daemon process ID and then sending it a Software Terminate signal (see the **signal** system call in *AIX Operating System Technical Reference*). If you do not specify *kernel-image*, **errstop** uses **/unix**. Only a user operating with superuser authority can run **errstop**.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Files

/unix	System kernel image.

Related Information

The following commands: "**errdemon**" on page 398 and "**ps**" on page 786.

The **kill** system call in *AIX Operating System Technical Reference.*

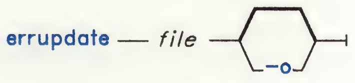
AIX Operating System Programming Tools and Interfaces.

errupdate

Purpose

Updates an error report template.

Syntax



OL805332

Description

The **errupdate** command adds, replaces, or deletes error report format templates in the file **/etc/errfmt**. **errupdate** creates an undo file in the current directory that it names *file*.**undo.err**. You can use this undo file as input to **errupdate** with the **-o** (override) flag to undo the changes **errupdate** has just made.

The **errupdate** command adds the extension **.err** to the *file* name you specify and reads update commands from the file with that name and extension. The first field of each template contains an operator:

+	To add or replace a template
-	To delete a template.

If the operation is +, then the following fields contain the template to be replaced. If the operation is a -, then the second field contains the class/subclass/mask identifier of the template to delete. **errupdate** checks for valid combinations of identifiers and writes error messages if it encounters invalid combinations. When adding or replacing, it compares the version numbers of each input template with the version number of the existing template of the same class/subclass/mask and, if the version number of the input template is later, replaces the old template with the input template. If the template does not already exist, then it is added to the file. The input template *must* contain an identifier line on the first line:

```
* /etc/errfmt
```

or **errupdate** rejects the input file. All delete operations are performed before the add/replace operations.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flag

-o	Does no version number checking.

Example

The following is an example input file:

```
* /etc/errfmt
+ H87 2.0 Native Serial: IODN D2: IOCN D2: Base_Addr D4:\
          Dev_Name A4: \n: Dev_Type X4: DDI_Length D4: Error _Type X1:\
          Last_I/O X1: Line_Status X1: Printer_Status X1:
- H92
```

Files

/etc/errfmt
file.err
file.undo.err

Related Information

The following command: "**errpt**, **errpd**" on page 400.

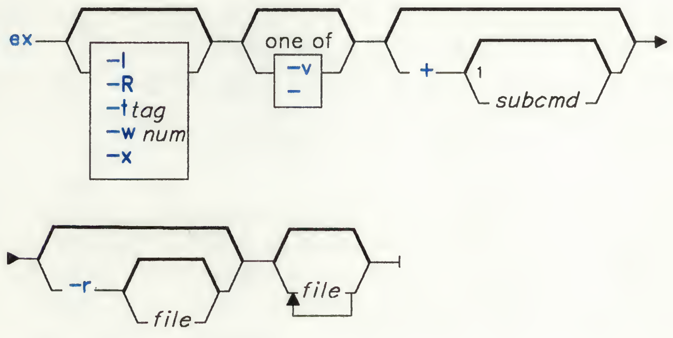
AIX Operating System Programming Tools and Interfaces.

ex

Purpose

Edits lines interactively, with screen display.

Syntax



OL805325

¹ Do not put a blank between these items.

OL805308

Description

The **ex** command is a line-oriented text editor that is a subset of the **vi** screen editor. The **ex** editor is similar to **ed**, but is more powerful, providing multiline displays and access to a screen editing mode. You may prefer to call **vi** directly to have environment variables set for screen editing. Also **edit**, a limited subset of **ex**, is available for novice or casual use. For more information on **vi**, see "**vi, vedit, view**" on page 1187. For more information on **edit**, see "**edit**" on page 387.

Notes:

1. Some **vi** subcommands have meanings that differ from **ed** subcommands.
2. To determine how to drive your work station more efficiently, **ex** uses the work station capability database **terminfo** and the type of the work station you are using from the shell environment variable **TERM**.

The **ex** editor has the following features:

- You can view text in files. The **z** subcommand lets you access windows of text, and you can scroll through text by pressing **Ctrl-D** and **Ctrl-U**. The **vi** subcommand provides further viewing options and active screen-editing by invoking the **vi** editor.
- You can revoke the last previous subcommand entered (except for **q** and **w**). The **undo** subcommand allows you to "undo" the last subcommand, even if it was an **undo** subcommand. Thus you can switch back and forth between the latest change in the edit file and the last prior file status and view the effect of a subcommand without that effect being permanent. The **ex** command displays changed lines and indicates when more than a few lines are affected by a subcommand. The **undo** subcommand causes all marks to be lost on lines changed and then restored if the marked lines were changed. It does not clear the `buffer modified` condition.
- You can retrieve your work (except changes that were in the buffer) if the system or the editor crashes by re-entering the editor with the **-r** flag and the file name. When the file name is not specified, all open files in your partition are listed.
- You can queue a sequence or group of files to edit. You can list the files in the **ex** command and then use the **next** subcommand to access each file sequentially. Or after you enter the editor, you can enter the **next** subcommand with a list of file names or a pattern (as used by the shell) to specify a set of files. In general, you can designate file names to the editor using the pattern-matching symbols that the shell will accept. You can use the wild card character % to form file names and represent the name of the current edit file.
- You can use a group of buffers (buffers named **a** through **z**) to move text between files and within a file. You can temporarily place text in these buffers and copy or reinsert it in a file, or you can carry it over to another file. The buffers are cleared when you quit the editor. The editor does not notify you if text is placed in a buffer and not used before exiting the editor.
- You can use patterns that match words. For example, you can search only for the word “ink” when your document also contains the word “inkblot” or “blink.”
- You can display a window of logical lines. The **z** subcommand allows you to select the number of lines displayed and locate the current line within the display simultaneously. More than a screen of output can result when the file lines are longer than the output display lines because the set number of logical lines are displayed rather than a number of physical lines.
- You can read a file of editor subcommands. The **so** command allows you to read a file of subcommands. Nesting of source files is permitted, allowing one file to call another; however, no return mechanism is provided.

The **ex** editor has the following maximum limits:

- 1024 characters per line
- 256 characters per global command list
- 128 characters in the previous inserted and deleted text
- 100 characters in a shell escape command
- 63 characters in a string-valued option
- 30 characters in a tag name
- 250,000 lines of 1024 characters per line silently enforced
- 32 map macros with 512 characters total.

Editing States

command	Normal and initial state. Input is prompted for by : (colon). Pressing **END OF FILE (Ctrl-D)** clears an uncompleted subcommand from the command line.
entry	Entered by **a, i** and **c**. In this state you can enter text. Entry state ends normally with a line that has only a . (period) on it or ends abnormally if you press **INTERRUPT (Alt-Pause)**.
visual	Entered by **vi, vi., vi-**, or **o**. Each of the first three commands gives you a full screen **vi** editor, but puts the current line in a different place on entry. Enter **vi** to put the current line at the top of the screen; enter **vi.** to put the current line in the middle of the screen; and enter **vi-** to put the current line at the bottom of the screen. The **o** command opens a one-line window. All three commands share the input state of the visual editor. Press the **Esc** key to exit the input state. To return to the **ex** command state at the current line, enter **Q** or ^\ while not in the input state.

Subcommands

The following table lists the **ex** subcommands. Most of these subcommands are discussed under "**edit**" on page 387 or "**vi, vedit, view**" on page 1187.

ab	abbrev	**l**	list	**rec**	recover	**x**	exit
a	append	**map**	map	**rew**	rewind	**ya**	yank
ar	args	**ma**	mark	**se**	set	**z**	window
c	change	**m**	move	**sh**	shell	**!**	escape
co	copy	**n**	next	**so**	source	**<**	lshift
d	delete	**nu**	number	**s**	substitute	**CR**	print next
e	edit	**pre**	preserve	**una**	unabbrev	**&**	resubst
f	file	**p**	print	**u**	undo	**>**	rshift
g	global	**pu**	put	**unm**	unmap	**^D**	scroll
i	insert	**q**	quit	**vi**	visual		
j	join	**re**	read	**w**	write		

Subcommand Addresses

$	The last line	*x-num*	The *num*th line before *x*
+	The next line	*x,y*	Lines *x* through *y*
−	The previous line	*'m*	The line marked with *m*
+ *num*	The *num*th line forward	''	The previous context
−*num*	The *num*th previous line	*/$pat*	The next line with *pat* at end of line
%	The first through last lines	*/^pat*	The next line with *pat* at start of line
num	line *num*	*/pat*	The next line with *pat*
.	The current line	*?pat*	The previous line with *pat*

Scanning Pattern Formation

^	The beginning of the line
$	The end of the line
.	Any character
\<	The beginning of the word
\>	The end of the word
[*string*]	Any character in *string*
[^*string*]	Any character not in *string*
[*x-y*]	Any character between *x* and *y*, inclusive
*****	Any number of the preceding character.

Flags

-l	Indents appropriately for Lisp code, and accepts the () {} [[and]] characters as text rather than interpreting them as **vi** subcommands. The *Lisp* modifier is active in **open** or **visual** modes.
-r [*file*]	Recovers *file* after an editor or system crash. If you do not specify *file*, a list of all saved files is displayed.
-R	The **readonly** option is set, preventing you from altering the file.
-t *tag*	Loads the file that contains *tag* and positions the editor at *tag*.
-v	Invokes the **visual** editor. **Note:** When the **v** flag is selected, an enlarged set of subcommands are available, including screen editing and cursor movement features. See "**vi, vedit, view**" on page 1187.
-	Suppresses all interactive-user feedback. If you use this flag, file input/output errors do not generate a helpful error message.

+*subcmd*	Begins edit at the specified editor search or subcommand. When *subcom*and is not entered, +places the current line to the bottom of the file. Normally **ex** sets current line to the start of the file, or to some specified tag or pattern.

Files

/usr/lib/exrecover	Recover subcommand.
/usr/lib/expreserve	Preserve subcommand.
/usr/lib/*/*	Describes capabilities of work stations.
$HOME/.exrc	Editor startup file.
./.exrc	Editor startup file.
/tmp/Ex*nnnnn*	Editor temporary.
/tmp/Rx*nnnnn*	Names buffer temporary.
/usr/preserve	Preservation directory.

Related Information

The following commands: "**vi**, **vedit**, **view**" on page 1187, "**edit**" on page 387, "**awk**" on page 81, "**ed**" on page 371, "**grep**" on page 501, and "**sed**" on page 887.

The **curses** subroutine and the **TERM**, **INIT**, and **terminfo** files in *AIX Operating System Technical Reference*.

expr

Purpose

Evaluates arguments as expressions.

Syntax

```
expr — expression —|
```

OL805048

Description

The **expr** command reads an *expression*, evaluates it, and writes the result to standard output. Within *expression*, you must separate each term with blanks, precede characters special to the shell with a backslash (\), and quote strings containing blanks or other special characters. Note that **expr** returns 0 to indicate a zero value, rather than the null string. Integers may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, twos complement numbers.

The operators and keywords are described in the following listing. Characters that need to be escaped are preceded by a backslash (\). The list is in order of increasing precedence, with equal precedence operators grouped within braces ({}).

expression1 \| *expression2*
: Returns *expression1* if it is neither null nor 0; otherwise it returns *expression2*.

expression1 \& *expression2*
: Returns *expression1* if neither *expression1* nor *expression2* is null or 0; otherwise it returns 0.

expression1 { =, \>, \>=, \<, \<=, != } *expression2*
: Returns the result of an integer comparison if both expressions are integers; otherwise returns the result of a string comparison.

expression1 {+, - } *expression2*
: Adds or subtracts integer-valued arguments.

expression1 { *, /, % } *expression2*
: Multiplies, divides, or provides the remainder from the division of integer-valued arguments.

expression1 : expression2

Compares *expression1* with *expression2*, which must be a pattern; pattern syntax is the same as that of the **ed** command (see page 371), except that all patterns are ***anchored***, so ^ (which anchors a pattern to the beginning of a line), is not a special character in this context.

Normally, the matching operator returns the number of characters matched. Alternatively, you can use the **\(. . . \)** symbols in *expression2* to return a portion of *expression1*. In an expression such as **[a-z]**, the minus means "through" according to the current collating sequence.

A collating sequence can define ***equivalence classes*** for use in character ranges. See "Overview of International Character Support" in *Managing the AIX Operating System* for more information on collating sequences and equivalence classes.

Japanese Language Support Information

A collating sequence does not define equivalence classes for use in character ranges. To avoid unpredictable results when using a range expression to match a class of characters, use a ***character class expression*** rather than a standard range expression. For information about character class expressions, see the discussion of this topic included in the description of the command "**ed**" on page 371.

The **expr** command returns the following exit values:

0	The expression is neither null nor 0.
1	The expression is null or 0.
2	The expression is invalid.

Note: After parameter processing by the shell, **expr** cannot distinguish between an operator and an operand except by the value. Thus, if $a is =, the command:

```
expr $a = '='
```

looks like:

```
expr = = =
```

after the shell passes the arguments to **expr**, and they will all be taken as the = operator. The following works:

```
expr X$a = X=
```

Examples

1. To modify a shell variable:

   ```
   COUNT=`expr $COUNT + 1`
   ```

   This adds 1 to the shell variable COUNT. The **expr** command is enclosed in grave accents, which causes the shell to substitute the standard output from **expr** into the COUNT= command. For more details, see "Command Substitution" on page 925.

2. To find the length of a shell variable:

   ```
   LENGTH=`expr $STR : ".*"`
   ```

   This sets LENGTH to the value given by the : (colon) operator. The pattern .* matches any string from beginning to end, so the colon operator gives the length of STR as the number of characters matched. Note that ".*" must be in quotes to prevent the shell from treating the * as a pattern-matching character. The quotes themselves are not part of the pattern.

   If STR is set to the null string, the error message `expr: syntax error` is displayed. This happens because the shell does not normally pass null strings to commands. In other words, the **expr** command sees only

   ```
   : .*
   ```

   (The shell also removes the quotation marks.) This does not work because the colon operator requires two values. This problem can be fixed by enclosing the shell variable in double quotation marks:

   ```
   LENGTH=`expr "$STR" : ".*"`
   ```

   Now if STR is null, LENGTH is set to zero. Enclosing shell variables in double quotation marks is recommended in general. However, do *not* enclose shell variables in single quotation marks. See page 918 for details about using quotation marks.

3. To use part of a string:

   ```
   FLAG=`expr "$FLAG" : "-*\(.*\)"`
   ```

   This removes leading minus signs, if any, from the shell variable FLAG. The colon operator gives the part of FLAG matched by the part of the pattern enclosed in \(\). If you omit the \(\), the colon operator gives the number of characters matched.

   If FLAG is set to - (minus), a syntax error message is displayed. This happens because the shell substitutes the value of FLAG before running the **expr** command. **expr** does not know that the minus is the value of a variable. It can only see:

   ```
   - : -*\(.*\)
   ```

and it interprets the first minus sign as the subtraction operator. We can fix this problem by using:

```
FLAG=expr "x$FLAG" : "x-*\(.*\)"
```

4. To use **expr** in an **if** statement:

   ```
   if expr "$ANSWER" : "[yY]" >/dev/null
   then
      # ANSWER begins with "y" or "Y"
   fi
   ```

   If ANSWER begins with y or Y, the **then** part of the **if** statement is performed. If the match succeeds, the result of the expression is 1 and **expr** returns an exit value of 0, which is recognized as the logical value TRUE by **if**. If the match fails, the result is 0 and the exit value 1 (FALSE).

   Redirecting the standard output of **expr** to the **/dev/null** special file discards the result of the expression. If you do not redirect it, the result is written to the standard output, which is usually your work station display.

5. Consider the following expression:

   ```
   expr "$STR" = "="
   ```

   If STR has the value = (equal sign), then after the shell processes this command **expr** sees the expression:

   ```
   = = =
   ```

   The **expr** command interprets this as three = operators in a row and displays a syntax error message. This happens whenever the value of a shell variable is the same as one of the **expr** operators. You can avoid this problem by doing the following:

   ```
   expr "x$STR" = "x="
   ```

Related Information

The following commands: "**ed**" on page 371 and "**sh**" on page 913.

"Overview of International Character Support" in *Managing the AIX Operating System.*

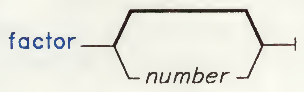
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

factor

Purpose

Factors a number.

Syntax



OL805051

Description

When called without an argument, the **factor** command waits for you to enter a positive number less than 2^{56}. It then writes the prime factors of that number to standard output. It displays each factor the proper number of times. To exit, enter a 0 or any nonnumeric character.

When called with an argument, **factor** determines the prime factors of *number*, writes the results to standard output, and exits.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Example

To calculate the prime factors of 123:

```
factor  123
```

This displays:

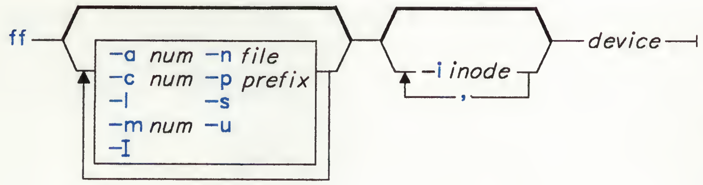
```
123
     3
     41
```

ff

Purpose

Lists the file names and statistics for a file system.

Syntax



OL805122

Description

Warning: This program is not intended for use with diskette-based file systems because of the difference in superblock structure and the general format of the file system.

The **ff** command reads the i-list and directories specified by *device* and writes information about them to standard output. It assumes that *device* is a file system, and saves i-node data for files specified by flags. The output from the **ff** command consists of the path name for each saved i-node, in addition to other file information that you request with the flags. The output is listed in order by i-node number, with tabs between all fields. The default line produced by **ff** includes the path name and i-number fields. With all flags enabled, the output fields include path name, i-number, size, and UID.

The *num* parameter in the flags descriptions is a decimal number, where +*num* means more than *num*, *-num* means less than *num*, and *num* means exactly *num*. A day is defined as a 24-hour period.

The **ff** command lists only a single path name out of many possible ones for an i-node with more than one link, unless you specify the **-l** flag. With **-l**, **ff** applies no selection criteria to the names listed. All possible names for every linked file on the file system are included in the output. On very large file systems, memory may run out before **ff** does.

Flags

-a *num*	Selects if the i-node has been accessed in *num* days.
-c *num*	Selects if the i-node has been changed in *num* days.
-i *i-node*	Generates names for only those i-nodes specified in the *inode* list.
-I	Does not display the i-node number after each path name.
-l	Generates a list of all path names for files with more than one link.
-m *num*	Selects if the file associated with the i-node has been modified in *num* days.
-n *file*	Selects if the file associated with the i-node has been modified more recently than the specified *file*.
-p *prefix*	Adds the specified *prefix* to each path name. The default prefix is **.** (dot).
-s	Writes the file size, in bytes, after each path name.
-u	Writes the owner's login name after each path name.

Examples

1. To list the path names of all files in a given file system:

   ```
   ff  -I  /dev/hd0
   ```

   This displays the path names of the files on the `/dev/hd0` disk. If you do not specify the `-I` flag, then **ff** also displays the i-number of each file.

2. To list files that have been modified recently:

   ```
   ff  -m  -2  -u  /dev/hd0
   ```

   This displays the path name, i-number, and owner's user name (`-u`) of each file on `/dev/hd0` that has been modified within the last two days (`-m -2`).

3. To list files that have *not* been used recently:

   ```
   ff  -a  +30  /dev/hd0
   ```

   This displays the path name and i-number of each file that was last accessed more than 30 days ago (`-a +30`).

4. To find out the path names of certain i-nodes:

   ```
   ff  -l  -i  451,76  /dev/hd0
   ```

   This displays all the path names (`-l`) associated with i-nodes `451` and `76`.

Related Information

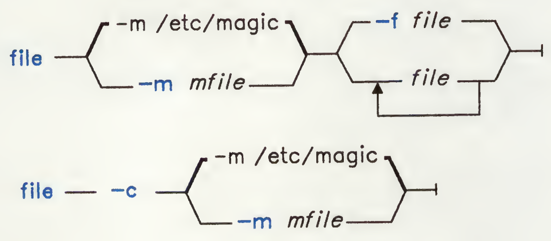
The following commands: "**find**" on page 422 and "**ncheck**" on page 683.

file

Purpose

Determines file type.

Syntax



OL805124

Description

The **file** command reads its input *files*, performs a series of tests on each one, and attempts to classify them by their types. The command then writes the file types to standard output.

If a file appears to be ASCII, **file** examines the first 512 bytes and tries to determine its language. If a file does not appear to be ASCII, **file** further attempts to distinguish a binary data file from a text file that contains extended characters.

If *file* is an **a.out** file, and the version number is greater than zero (see "**ld**" on page 557), **file** displays the version stamp.

The **file** command uses the file **/etc/magic** to identify files that have a ***magic number***, that is, any file containing a numeric or string constant that indicates its type. Comments at the beginning of **/etc/magic** explain its format.

Japanese Language Support Information

The **file** command uses the **magic.cat** message catalog. If **magic.cat** cannot be opened, the **file** command uses the **/etc/magic** file mentioned earlier.

Flags

-c	Checks the *mfile* (**/etc/magic** by default) for format errors. This validation is not normally done. File typing is not done under this flag.
-f *file*	Reads *file* for a list of files to examine.
-m *mfile*	Specifies *mfile* as the magic file (**/etc/magic** by default).

Examples

1. To display the type of information a file contains:

   ```
   file  myfile
   ```

   This displays the file type of `myfile` (directory, data, ASCII text, C-program source, archive, and so forth).

2. To display the type of each file named in a list of file names:

   ```
   file  -f  filenames
   ```

   This displays the type of each file with a name that appears in `filenames`. Each file name must appear alone on a line.

   To create `filenames`:

   ```
   ls  >filenames
   ```

   then edit `filenames` as desired.

Files

/etc/magic	File type database.

Related Information

"Overview of International Character Support" in *Managing the AIX Operating System.*

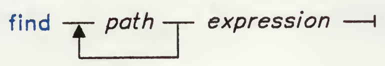
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

find

Purpose

Finds files matching expression.

Syntax



OL805125

Description

The **find** command recursively searches the directory tree for each specified *path*, seeking files that match a Boolean *expression* written using the terms given below. The output from **find** depends on the terms used in *expression*.

Expression Terms

In the following descriptions, the parameter *num* is a decimal integer that can be specified as +*num* (more than *num*), -*num* (less than *num*), or *num* (exactly *num*).

-inum *n*	True if file has inode *n*.
-name *file*	True if *file* matches the file name. You can use pattern-matching characters, provided they are quoted. In an expression such as **[a-z]**, the minus means "through" according to the current collating sequence. A collating sequence may define ***equivalence classes*** for use in character ranges. See "Overview of International Character Support" in *Managing the AIX Operating System* for more information on collating sequences and equivalence classes.

Japanese Language Support Information

A collating sequence in Japanese Language Support does not define equivalence classes for use in character ranges. To avoid unpredictable results when using a range expression to match a class of characters, use a ***character class expression*** rather than a standard range expression. For information about character class expressions, see the discussion in "File Name Substitution" on page 4.

-node *nname*	True if the file resides in the node *nname* where the nodes are connected through Distributed Services. If *nname* is a valid nickname, it is used as is. If *nname* is not a valid nickname but has a valid NID syntax, it is used as a NID.
-perm *onum*	True if the file permission code of the file exactly matches the octal number *onum* (see "**chmod**" on page 160 for an explanation of file permissions). The *onum* parameter may be up to three octal digits. If you want to test the higher-order permission bits (the set-user-ID bit or set-group-ID bit, for example), prefix the *onum* parameter with a minus (-) sign. This makes more flag bits significant (see the **stat** system call for an explanation of the additional bits), and also changes the comparison to: (*flags*&*onum*)= =*onum*
-type *type*	True if the file *type* is of the specified type as follows: **b** Block special file; **c** Character special file; **d** Directory; **f** Plain file; **p** FIFO (a named pipe).
-links *num*	True if the file has *num* links. See "**ln**" on page 581.
-user *uname*	True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the **/etc/passwd** file, it is interpreted as a user ID.
-group *gname*	True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the **/etc/group** file, it is interpreted as a group ID.
-size *num*	True if the file is *num* blocks long (512 bytes per block). For this comparison the file size is rounded up to the nearest block.
-atime *num*	True if the file has been accessed in *num* days.
-mtime *num*	True if the file has been modified in *num* days.

-ctime *num*	True if the file i-node has been changed in *num* days.
-exec *cmd*	True if the *cmd* runs and returns a zero value as exit status. The end of *cmd* must be punctuated by a quoted or escaped semicolon. A command parameter {} is replaced by the current path name.
-ok *cmd*	The **find** command asks you whether it should start *cmd*. If your response begins with `y`, *cmd* is started. The end of *cmd* must be punctuated by a quoted or escaped semicolon.
-print	Always true; causes the current path name to be displayed. **find** does not display path names unless you specify this expression term.
-cpio *device*	Write the current file to *device* in **cpio** format. See "**cpio**" on page 205.
-newer *file*	True if the current file has been modified more recently than the file indicated by *file*.
-depth	Always true. This causes the descent of the directory hierarchy to be done so that all entries in a directory are affected before the directory itself. This can be useful when **find** is used with **cpio** to transfer files that are contained in directories without write permission.
\\(*expression* **\\)**	True if the expression in parentheses is true.

You may perform the following logical operations on these terms (listed in order of decreasing precedence):

- Negate a term (! is the NOT operator).
- Concatenate terms (juxtaposing two terms implies the AND operation).
- Alternate terms (**-o** is the OR operator).

Examples

1. To list all files in the file system with a given base file name:

   ```
   find  /  -name  .profile  -print
   ```

   This searches the entire file system and writes the complete path names of all files named `.profile`. The / tells **find** to search the root directory and all of its subdirectories. This may take a while, so it is best to limit the search by specifying the directories where you think the files might be.

2. To list the files with a specific permission code in the current directory tree:

   ```
   find  .  -perm  0600  -print
   ```

   This lists the names of the files that have *only* owner-read and owner-write permission. The **.** (dot) tells **find** to search the current directory and its subdirectories. See "**chmod**" on page 160 for details about permission codes.

3. To search several directories for files with certain permission codes:

   ```
   find  manual  clients  proposals  -perm  -0600  -print
   ```

   This lists the names of the files that have owner-read and owner-write permission *and possibly other permissions*. The directories `manual`, `clients`, and `proposals`, and their subdirectories, are searched. Note that `-perm 0600` in the previous example selects only files with permission codes that match `0600` *exactly*. In this example, `-perm -0600` selects files with permission codes that allow *at least* the accesses indicated by `0600`. This also matches the permission codes `0622` and `2744`.

4. To search for regular files with multiple links:

   ```
   find  .  -type  f  -links  +1  -print
   ```

   This lists the names of the ordinary files (`-type f`) that have more than one link (`-links +1`). Note that every directory has at least two links: the entry in its parent directory and its own . (dot) entry. See "**ln**" on page 581 for details about multiple file links.

5. To back up selected files in **cpio** format:

   ```
   find  .  -name  "*.c"  -cpio  /dev/rfd0
   ```

   This saves all the `.c` files onto the diskette in **cpio** format. See "**cpio**" on page 205 for details. Note that the pattern `"*.c"` must be quoted to prevent the shell from treating the * as a pattern-matching character. This is a special case in which **find** itself decodes the pattern-matching characters.

6. To perform an action on all files that meet complex requirements:

   ```
   find . \( -name a.out -o -name "*.o" \) -atime +7 -exec rm {} \;
   ```

   This deletes (`-exec rm {} \;`) all files named `a.out` or that end with `.o`, and that were last accessed over seven days ago (`-atime +7`). The `-o` flag is the logical OR operator.

Files

/etc/group	File that contains all known groups.
/etc/passwd	File that contains all known users.

Related Information

The following commands: "**cpio**" on page 205, "**sh**" on page 913, and "**test**" on page 1064.

The **stat** system call and the **cpio** and **fs** files in *AIX Operating System Technical Reference.*

"Overview of International Character Support" and "Using Distributed Services" in *Managing the AIX Operating System.*

The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

fish

Purpose

Plays the card game Go Fish.

Syntax

```
/usr/games/fish —┤
```

OL805189

Description

The object of the **fish** game is to accumulate ***books*** of four cards with the same face value. You and the program take turns asking each other for a card in your hand. If your opponent has one or more cards of that value, he must hand them over. If not, he says `GO FISH`, and you draw a card from the pool of undealt cards. If you draw the card you asked for, you draw again. As books are made, they are laid down on the table. Play continues until there are no cards left. The player with the largest number of books wins the game. **fish** tells you the winner and exits.

The **fish** game asks if you want instructions before play begins. To see the instructions, enter `y` or `yes`.

Entering **p** as your first move gives you the professional-level game.

The **fish** game tells you the cards in your hand each time it prompts for a move. It tells you when either side makes a book, says `GO FISH` for you, and draws for you. All you must enter as play progresses is the value of the card you want to ask for. If you press only the **Enter** key, you are given information about the number of cards in your opponent's hand and in the pool.

To exit the game before play is completed, press INTERRUPT (**Alt-Pause**).

fmt

Purpose

Formats mail messages prior to sending.

Syntax

```
/usr/bin/fmt ── file ──┤
```

AJ2FL123

Description

The **fmt** command invokes a simple text formatter that reads the concatenation of input *files* (or standard input if no *files* are specified). It then produces, on standard output, a version of the input with lines as close to 72 characters long as possible. The spacing at the beginning of the input lines is preserved in the output, as are blank lines and interword spacing.

The **fmt** command is generally used to format mail messages prior to sending them through the mail facility. It may also be useful, however, for other simple formatting tasks. For example, within visual mode of a text editing program such as **vi**, the command `!}fmt` reformats a paragraph so that all lines are approximately 72 characters long.

Note: The **fmt** command is a fast, simple formatting program. Standard text editing programs are more appropriate than **fmt** for complex formatting operations.

Related Information

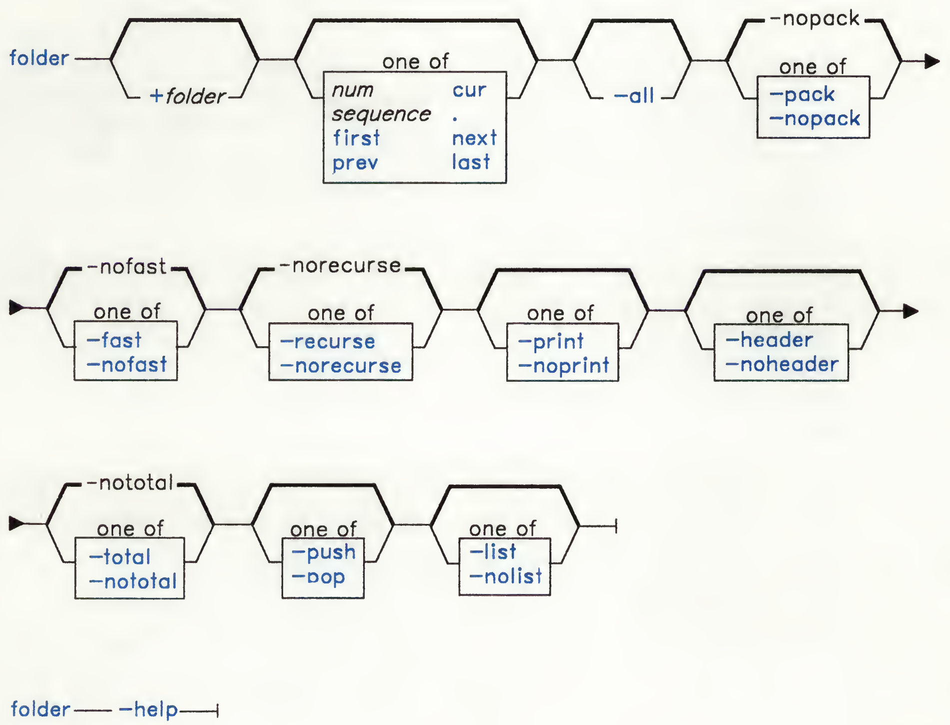
The following commands: "**mail, Mail**" on page 608, and "**nroff, troff**" on page 709.

folder

Purpose

Selects and lists folders and messages.

Syntax



AJ2FL228

Description

The **folder** command is used to set the current folder and the current message for that folder, and to list information about your folders. The **folder** command is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

The **folder** command entered with only the **-all** flag gives a line of information for each folder in your mail directory. This information tells how many messages are in each folder, what the current message is for each folder, what the range of message numbers is in each folder, and if a folder is the current folder. The **folder** command specified without arguments provides information for the current folder. The **-recurse** flag displays this information for all folders and subfolders in your entire mail directory structure.

Specify a folder with the **folder** command to make it the current folder, and a message to make it the current message for that folder. You can also create a folder-stack to manipulate a group of folders by using the **-push**, **-pop**, and **-list** flags.

Flags

-all	Displays a line of information about each folder in your mail directory. Specify *+folder* to display the information about that folder and the subfolders within that folder's directory. Add the **-recurse** flag to display the information about the specified folder and for all subfolders in all directories under the specified folder's directory.
-fast	Displays only the names of the folders.
+folder msg	Sets the specified folder as the current folder, and the specified message as the current message. You can use the following message references when specifying *msgs*:

num	*sequence*	**first**
prev	**cur**	**.**
next	**last**	

If you specify a *sequence*, that sequence must contain one message only.

The default folder is the current folder. The default message is none.

-header	Displays column headings for the folder information.
-help	Displays help information for the command.
-list	Displays the current folder followed by the contents of the folder-stack.
-nofast	Displays information about each folder. This flag is the default.
-noheader	Suppresses column headings for the folder information. This flag is the default.
-nolist	Suppresses the display of the folder-stack contents. This flag is the default.
-nopack	Does not renumber the messages in the folder. This flag is the default.
-noprint	Does not display folder information. If **-push**, **-pop**, or **-list** is specified, **-noprint** is the default.

-norecurse	Displays information about the top-level folders in your current folder only, not about subfolders. This flag is the default.
-nototal	Does not display a total of all messages and folders in your mail directory structure. **-total** is the default when **-all** is specified, otherwise **-nototal** is the default.
-pack	Renumbers the messages in the specified folder. This renumbering eliminates the gaps in the message numbering after messages have been deleted.
-pop	Removes the folder from the top of the folder-stack and makes it the current folder. +*folder* cannot be specified with the **-pop** flag.
-print	Displays information about the folders. This information includes the number of messages in each folder, the current message for each folder, and the current folder. If **-push**, **-pop**, or **-list** is specified, **-noprint** is the default, otherwise **-print** is the default.
-push	Moves the current folder to the top of the folder-stack and sets the specified folder as the current folder. If no folder is specified, **-push** swaps the current folder with the folder on the top of the folder-stack.
-recurse	Displays information about all folders and subfolders in your current folder. You can specify a folder to display the information about that folder and its subfolders only.
-total	Displays a total of all messages and folders in your mail directory structure. **-total** does not display information for subfolders unless you also specify the **-recurse** flag. The **-total** flag is the default if **-all** is specified.

Profile Entries

Current-Folder:	Sets your default current folder.
Folder-Protect:	Sets the protection level for your new folder directories.
Folder-Stack:	Specifies your folder stack.
lsproc:	Specifies the program used to list the contents of a folder.
Path:	Specifies your *user_mh_directory*.

Files

$HOME/.mh_profile	The MH user profile.

Related Information

Other MH commands: "**folders**" on page 433, "**mhpath**" on page 648, "**packf**" on page 733, "**refile**" on page 817.

The **mh-profile** file in *AIX Operating System Technical Reference.*

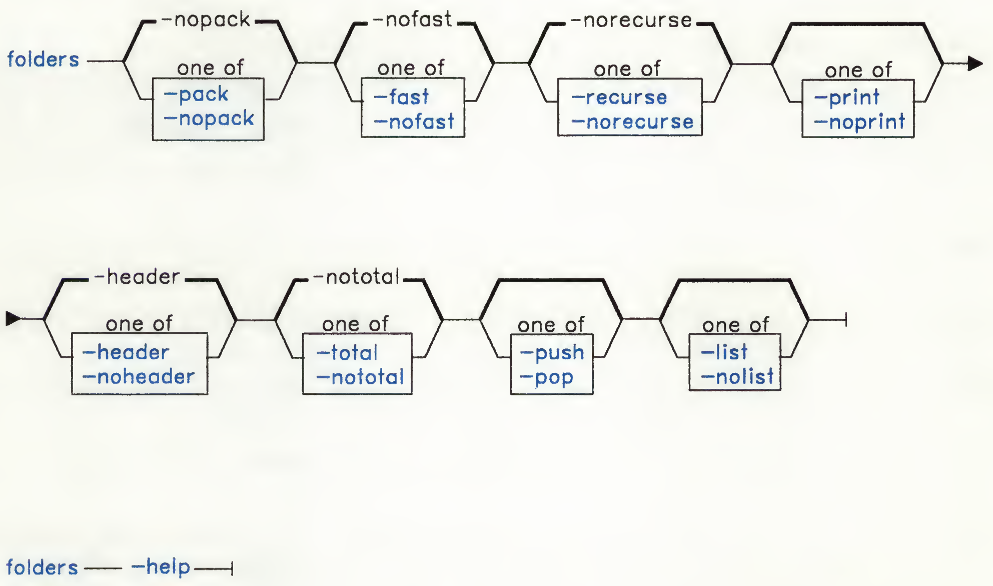
"Overview of the Message Handling Package" in *Managing the AIX Operating System.*

folders

Purpose

Lists folders and messages.

Syntax



AJ2FL229

Description

The **folders** command is used to list information about your folders. The **folders** command is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

The **folders** command is equivalent to the **folder** command specified with the **-all** flag.

Flags

-fast	Displays only the names of the folders in your mail directory.
-header	Displays column headings for the folder information. This flag is the default.
-help	Displays help information for the command.
-list	Displays the current folder followed by the contents of the folder-stack.
-nofast	Displays information about each folder in your mail directory. This flag is the default.
-noheader	Suppresses column headings for the folder information.
-nolist	Suppresses the display of the folder-stack contents. This flag is the default.
-nopack	Does not renumber the messages in the folder. This flag is the default.
-noprint	Does not display folder information. If **-push**, **-pop**, or **-list** is specified, **-noprint** is the default.
-norecurse	Displays information about the folders in your mail directory only, not about subfolders. This flag is the default.
-nototal	Does not display a total of all messages and folders in your mail directory structure.
-pack	Renumbers the messages in the folders. This eliminates the gaps in the message numbering after messages have been deleted.
-pop	Removes the folder from the top of the folder-stack and makes it the current folder.
-print	Displays information about the folders. This information includes the number of messages in each folder, the current message for each folder, and the current folder. If **-push**, **-pop**, or **-list** is specified, **-noprint** is the default, otherwise **-print** is the default.
-push	Swaps the current folder with the folder on the top of the folder-stack.
-recurse	Displays information about all folders and subfolders in your entire mail directory structure.
-total	Displays a total of all messages and folders in your mail directory structure. **-total** does not display information for subfolders unless you also specify the **-recurse** flag. The **-total** flag is the default.

Profile Entries

Current-Folder:	Sets your default current folder.
Folder-Protect:	Sets the protection level for your new folder directories.
Folder-Stack:	Specifies your folder stack.
lsproc:	Specifies the program used to list the contents of a folder.
Path:	Specifies your *user_mh_directory*.

Files

$HOME/.mh_profile	The MH user profile.

Related Information

Other MH commands: "**folder**" on page 429, "**mhpath**" on page 648, "**packf**" on page 733, "**refile**" on page 817.

The **mh-profile** file in *AIX Operating System Technical Reference*.

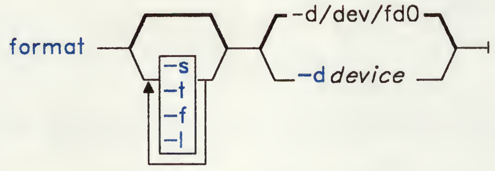
"Overview of the Message Handling Package" in *Managing the AIX Operating System*.

format

Purpose

Formats diskettes.

Syntax



OL805395

Description

The **format** command formats diskettes in the specified *device* (**/dev/fd0** by default). **format** determines the device type, either a 360K or a 1.2M diskette drive. By default, it formats a diskette in a 360K drive to have 40 cylinders, 9 sectors per track, and 2 sides and a diskette in a 1.2M drive to have 80 cylinders, 15 sectors per track, and 2 sides.

Flags

-d*device*	Specifies the device containing the diskette to be formatted.
-f	Formats the diskette without checking for bad tracks, thus formatting the diskette faster.
-l	Formats a 360K diskette in a 1.2M diskette drive. **Warning:** A 360K diskette drive may not be able to read a 360K diskette that has been formatted in a 1.2M drive.
-s	Specifies a single-sided diskette. Use only for 360K diskette drives.
-t	Specifies that the number of sectors on a 360K diskette should be 8.

Related Information

The **fd** file in *AIX Operating System Technical Reference.*

fortune

Purpose

Tells a fortune.

Syntax

```
/usr/games/fortune —|
```

OL805190

Description

The **fortune** game tells a fortune, selected at random from the file **/usr/games/lib/fortunes**, and exits.

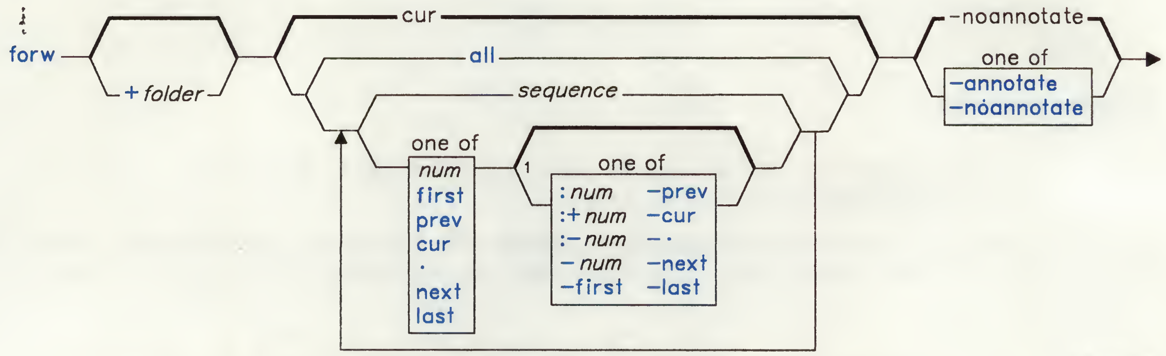
You can edit the file **/usr/games/lib/fortunes** to add your own fortunes. Each saying in the file should be a single line. **fortune** folds long sayings into multiple lines as necessary.

forw

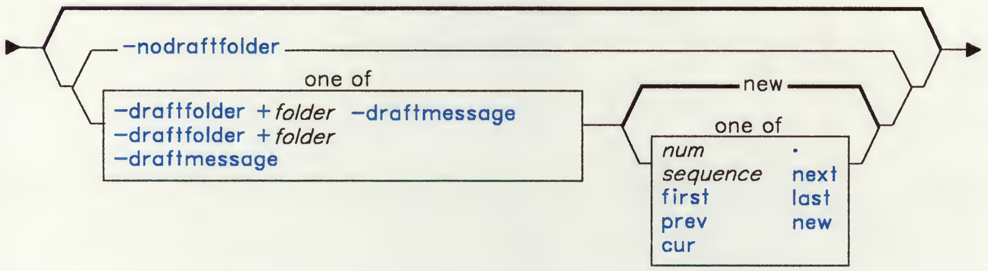
Purpose

Forwards messages.

Syntax



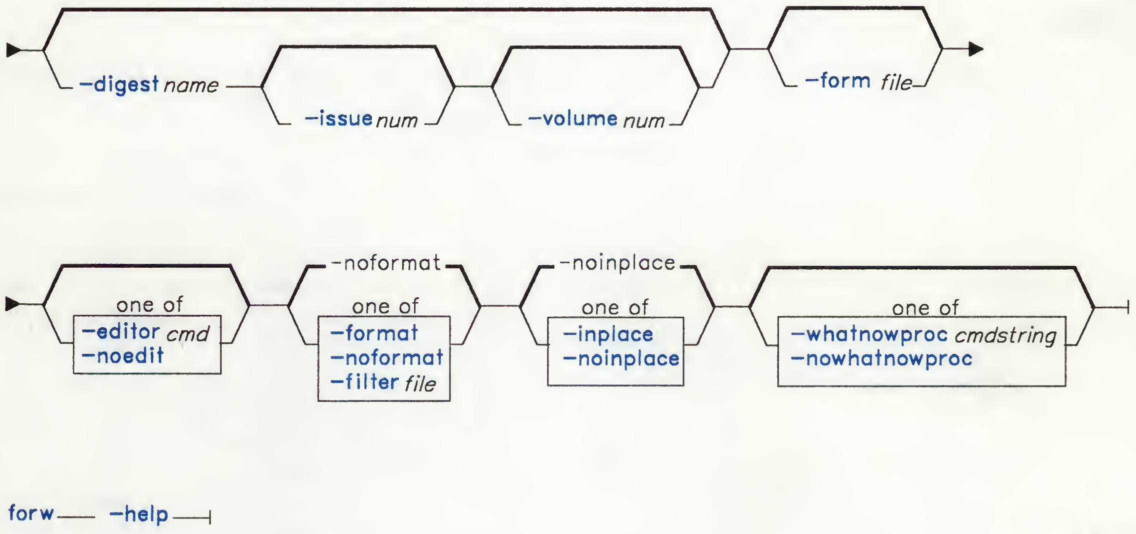
AJ2FL218



AJ2FL157

[1] Do not put a blank between these items.

OL805308



AJ2FL219

Description

The **forw** command is used to create a message containing other messages. **forw** is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

By default, **forw** copies a message form to a new draft message and invokes an editor. You can then fill in the message header fields **To:** and **Subject:** and fill in or delete the other header fields (such as **cc:** and **Bcc:**). When you exit the editor, the **forw** command invokes the MH command **whatnow**. You can press **Enter** to see a list of the available **whatnow** subcommands. These subcommands enable you to continue editing the message, list the message, direct the disposition of the message, or end the processing of the **forw** command. See "**whatnow**" on page 1215 for a description of the subcommands.

You can specify the messages that you want to distribute by using the *+folder msgs* flag. If you do not specify a message, **forw** forwards the current message.

You can specify the format of the message by using the **-form** flag. If you do not specify this flag, **forw** uses your default message format located in the file *user_mh_directory*/**forwcomps**. If this file does not exist, **forw** uses the system default message format located in **/usr/lib/mh/forwcomps**.

Note: The line of dashes or a blank line must be left between the header and the body of the message for the message to be identified when it is sent.

Flags

-annotate	Annotates the messages being forwarded with the lines: `Forwarded:` *date* / `Forwarded:` *addrs* The annotation appears in the original draft message so that you can maintain a complete list of recipients with the original message. If you do not actually redistribute the message using the immediate **forw** command, the **-annotate** may fail to provide annotation. The **-inplace** flag forces annotation to be done in place.
-digest *name*	Uses the digest facility to create a new issue for the digest called *name*. **forw** expands the format strings in the *components* file (using the same format string mechanism used by the **repl** command) and composes the draft using the standard digest encapsulation algorithm. After the draft has been composed, **forw** writes out the volume and issue entries for the digest and invokes the editor.
-draftfolder +*folder*	Places the draft message in the specified folder. If you do not specify this flag, **forw** selects a default draft folder according to the information supplied in the MH profiles. You can define a default draft folder in **$HOME/.mh_profile**. If **-draftfolder** +*folder* is followed by *msg*, *msg* represents the **-draftmessage** attribute.
-draftmessage *msg*	Specifies the draft message. You can specify one of the following message references as *msg*: *num*, *sequence*, **first**, **prev**, **cur**, **.**, **next**, **last**, **new**. The default draft message is **new**.
-editor *cmd*	Specifies that *cmd* is the initial editor for preparing the message. If you do not specify this flag, **forw** selects a default editor or suppresses the initial edit, according to the information supplied in the MH profiles. You can define a default initial editor in **$HOME/.mh_profile**.
-filter *file*	Reformats each message being forwarded and places the reformatted message in the draft message. **-filter** uses the **mhl** command and the specified format file. When you also specify the **-digest** flag, you may want to use the filter file **/usr/lib/mh/mhl.digest**.

+*folder msgs*	Specifies the messages that you want to forward. *msgs* can be several messages, a range of messages, or a single message. You can use the following message references when specifying *msgs*: *num* **first** **prev** **cur** **.** **next** **last** **all** *sequence* The default message is the current message in the current folder. If you specify several messages, the first message forwarded becomes the current message. If you specify a folder, that folder becomes the current folder.
-form *file*	Uses the form contained in the specified file to construct the beginning of the message. **forw** treats each line in *file* as a format string. If the **-digest** flag is also specified, **forw** uses the form specified in *file* for the format of the digest. If you do not specify a form for a digest, **forw** uses the format in the file *user_mh_directory*/**digestcomps**. If this file does not exist, **forw** uses the system default digest form specified in the file **/usr/lib/mh/digestcomps**.
-format	Reformats each message being forwarded and places the reformatted message in the draft message. **-format** uses the **mhl** command and a default format file. If *user_mh_directory*/**mhl.forward** exists, it contains the default format. Otherwise, **/usr/lib/mh/mhl.forw** contains the default format.
-help	Displays help information for the command.
-inplace	Forces annotation to be done in place in order to preserve links to the annotated message.
-issue *num*	Specifies the issue number of the digest. The default issue number is one greater than current value of the *digest_name*-**issue-list** entry in *user_mh_directory*/**context**.
-noannotate	Does not annotate the message. This flag is the default.
-nodraftfolder	Places the draft in the file *user_mh_directory*/**draft**.
-noedit	Suppresses the initial edit.
-noformat	Does not reformat the messages being forwarded. This flag is the default.
-noinplace	Does not perform annotation in place. This flag is the default.

-nowhatnowproc	Does not invoke a program that guides you through the forwarding tasks. The **-nowhatnowproc** flag also prevents any edit from occurring.
-volume *num*	Specifies the volume number of the digest. The default volume number is the current value of the *digest_name*-**volume-list** entry in *user_mh_directory*/**context**.
-whatnowproc *cmdstring*	Invokes *cmdstring* as the program to guide you through the forwarding tasks. See "**whatnow**" on page 1215 for information about the default **whatnow** program and its subcommands. **Note:** If you specify `whatnow` for *cmdstring*, **forw** invokes an internal **whatnow** procedure rather than a program with the file name **whatnow**.

Profile Entries

Current-Folder:	Sets your default current folder.
Draft-Folder:	Sets your default folder for drafts.
Editor:	Sets your default initial editor.
fileproc:	Specifies the program used to refile messages.
mhlproc:	Specifies the program used to filter messages being forwarded.
Msg-Protect:	Sets the protection level for your new message files.
Path:	Specifies your *user_mh_directory*.
whatnowproc:	Specifies the program used to prompt `What now?` questions.

Files

/usr/lib/mh/forwcomps	The MH default message skeleton.
user_mh_directory/forwcomps	The user's default message skeleton. (If it exists, it overrides the MH default message skeleton.)
/usr/lib/mh/digestcomps	The MH default message skeleton when **-digest** is specified.
user_mh_directory/digestcomps	The user's default message skeleton when **-digest** is specified. (If it exists, it overrides the MH default message skeleton.)
/usr/lib/mh/mhl.forward	The default MH message filter.
user_mh_directory/mhl.forward	The user's default message filter. (If it exists, it overrides the MH default message filter.)
$HOME/.mh_profile	The MH user profile.
user_mh_directory/draft	The draft file.
user_mh_directory/context	The context file.

Related Information

Other MH commands: "**ali**" on page 48, "**anno**" on page 50, "**comp**" on page 185, "**dist**" on page 336, "**dp**" on page 352, "**inc**" on page 518, "**mhl**" on page 643, "**msh**" on page 677, "**repl**" on page 821, "**send**" on page 893, "**whatnow**" on page 1215, "**whom**" on page 1222.

The **mh-alias**, **mh-format**, **mh-mail**, and **mh-profile** files in *AIX Operating System Technical Reference.*

"Overview of the Message Handling Package" in *Managing the AIX Operating System.*

fptype

Purpose

Displays the floating point configuration of the system.

Syntax

```
fptype ──┤
```

OL805471

Description

The **fptype** command calls the subroutine, **_FPtype**, which determines the current floating point configuration. Then **fptype** returns one of the following values and displays the corresponding message:

Value	Message
0	Floating Point Type = Software Emulation
1	Floating Point Type = FPA Card
2	Floating Point Type = APC Card with MC68881
4	Floating Point Type = AFPA - No DMA Support
12	Floating Point Type = AFPA - With DMA Support

Related Information

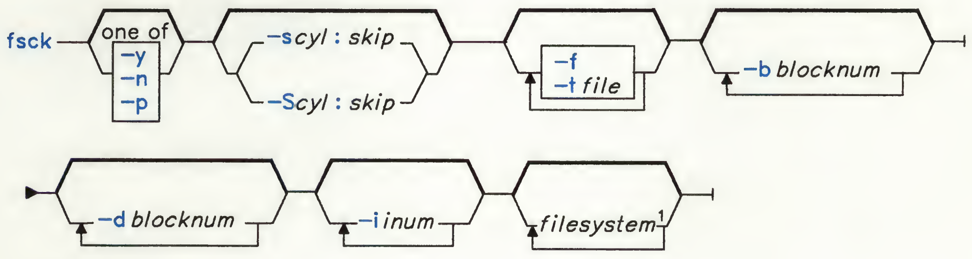
The **fpfp** subroutine in *AIX Operating System Technical Reference.*

fsck, dfsck

Purpose

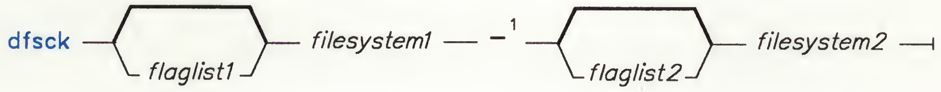
Checks file system consistency and interactively repairs the file system.

Syntax



¹The default action is to check every file system with the attribute check=true in the file /etc/filesystem.

OL8055384



¹Use a -- to separate the groups when you specify flags as part of the argument.

OL805456

Description

Warning: Always run **fsck** on file systems after a system crash. Corrective actions may result in some loss of data. The default action for each consistency correction is to wait for the operator to respond yes or no. If you do not have write permission for an affected file, **fsck** defaults to a no response in spite of your actual response.

The **fsck** command checks and interactively repairs inconsistent *filesystems*. It should be run on every file system as part of system initialization (see "**rc**" on page 806). You must have superuser authority to run **fsck**. Normally, the file system is consistent, and **fsck** merely reports on the number of files, used blocks and free blocks in the file system. If the *filesystem* is inconsistent, **fsck** displays information about the inconsistencies found and prompts you for permission to repair them. **fsck** is conservative in its repair efforts and tries to avoid actions that might result in the loss of valid data. In certain cases, however, **fsck** recommends the destruction of a damaged file.

If you do not specify *filesystem*, **fsck** looks at **/etc/filesystems** to find a list of file systems to check by default. **fsck** can perform checks (on separate arms) in parallel (running in parallel processes). This can reduce the time required to check a large number of file systems.

In **/etc/filesystems**, automatic checking may be enabled by adding a line in the stanza, as follows:

```
check=true
```

If you specify the **-p** flag, **fsck** can perform multiple checks at the same time. To tell **fsck** which file systems are on the same drives, change the **check** specification in **/etc/filesystems** as follows:

`check=`*number*

The *number* tells **fsck** which group contains a particular file system. File systems on a single drive are placed in the same group. Each group is checked in a separate parallel process. File systems are checked, one at a time, in the order that they appear in **/etc/filesystems**. All `check=true` file systems are in group 1. **fsck** attempts to check the root file system before any other file system regardless of order specified on the command line or in **/etc/filesystems**.

The **fsck** command checks for the following inconsistencies:

- Blocks allocated to multiple files or to a file and the free list.
- Blocks allocated to a file or on the free list outside the range allowable block numbers.
- Discrepancies between the number of directory references to a file and the link count in the file.
- Size checks:
  - Incorrect number of blocks.
  - Directory size not 16-byte aligned.
- Bad i-node format.
- Blocks not accounted for anywhere.
- Directory checks:
  - File pointing to an i-node that is not allocated.
  - I-node number out of range.
  - Dot (.) link missing or not pointing to itself.
  - Dot dot (..) link missing or not pointing to the parent directory.
  - Files that are not referenced or directories that are not reachable.

- Superblock checks:
  - More than 65535 i-nodes.
  - More blocks for i-nodes than there are in the file system.
- Bad free block list format.
- Total free block and/or free i-node count incorrect.

Orphaned files and directories (those that cannot be reached) are, if you allow it, reconnected by placing them in the **lost+found** subdirectory in the root directory. The name assigned is the i-node number. The only restriction is that the directory **lost+found** must already exist in the root directory of the file system being checked and must have empty slots in which entries can be made (accomplished by copying a number of files to the directory and then removing them before you run **fsck**). If you do not allow **fsck** to reattach an orphaned file, it requests permission to destroy the file. When **fsck** displays i-node information, the **NLTIME** environment variable controls the format of the modification time.

In addition to its messages, **fsck** records the outcome of its checks and repairs through its exit value. This exit value can be any sum of the following conditions:

0	All checked file systems are now okay.
2	**fsck** was interrupted before it could complete checks or repairs.
4	**fsck** changed the mounted file system; the user must restart the system immediately.
8	The file system contains unrepaired damage.

When the system is being started up normally, **fsck** runs with the **-p** flag from **/etc/rc** (see "**rc**" on page 806). If **fsck** detects and repairs errors on the root or other mounted file systems, it displays a message on the console and restarts the system, if possible. If it cannot restart the system or if it detects errors that it cannot repair, it displays appropriate messages on the console and returns an exit value indicating that an immediate restart is necessary.

Note: All statistics reported by **fsck** are in 512-byte blocks, regardless of the actual block size of the file system being checked. All user specifications should be specified in 512-byte blocks.

dfsck

The **dfsck** command lets you simultaneously check two file systems on two different drives. Use the *flaglist1* and *flaglist2* arguments to pass flags and parameters for the two sets of file systems. Use a - (minus) to separate the file system groups if you specify flags as part of the arguments.

The **dfsck** command permits you to interact with two **fsck** commands at once. To aid in this, **dfsck** displays the file system name with each message. When responding to a question from **dfsck**, you must prefix your response with a **1** or a **2** to indicate whether the answer refers to the first or second file system group.

Warning: Do not use **dfsck** to check the root file system (**/dev/hd0**).

Flags

-b*blocknum*	Designates a block as bad. **fsck** searches for any files that contain the specified block. If it finds any such files, it asks permission to delete them. If it finds no such files or is told to delete all such files, the specified block is added to the bad block list in i-node 1. This keeps the block out of circulation so that it cannot be allocated to any user file.
-d*blocknum*	Searches for references to a specified disk block. Whenever **fsck** encounters a file that contains a specified block, it displays the i-node number and all path names that refer to it.
-f	Performs a fast check. Under normal circumstances, the only file systems likely to be affected by halting the system without shutting down properly are those that were mounted when the system stopped. The **-f** flag tells **fsck** not to check file systems that were cleanly unmounted. The **fsck** command determines this by inspecting the **s_fmod** flag in the file system superblock. This flag is set whenever a file system is mounted and cleared when it is cleanly unmounted. If a file system was cleanly unmounted, it is unlikely to have any problems. Because most file systems are cleanly unmounted, not checking those file systems can reduce the checking time.
-i*inum*	Searches for references to a specified i-node. Whenever **fsck** encounters a directory reference to a specified i-node number, it displays the full path name of the reference.
-n	Assumes a no response to all questions asked by **fsck**; does not open *filesystem* for writing.
-p	Does not display messages about minor problems, but fixes them automatically. This flag does not grant the wholesale license that the **-y** flag does and is useful for performing automatic checks when the system is to be started normally. You should use this flag whenever the system is being run automatically as part of the system startup procedures.
-s[*cyl*:*skip*]	Ignores the actual free list and unconditionally reconstructs a new one. You can specify an optional interleave specification with this flag: *cyl* specifies the number of blocks per cylinder; *skip* specifies the number of blocks to skip. If you do not specify *cyl* or *skip*, **fsck** uses the interleave parameters in the superblock. The file system should be unmounted while this is done; if this is not possible, be sure that you are running no programs and that you perform a system restart immediately afterwards so that the old copy of the superblock in memory is not written to disk.
-S[*cyl*:*skip*]	Conditionally reconstructs the free list. This flag is like the **-s** flag except that the free list is rebuilt only if there are no discrepancies discovered in the file system. Using **-S** forces a no response to all questions asked by **fsck**. Use this flag to force free list reorganization on uncontaminated file systems.

-t*file*	Uses *file* as a scratch file if **fsck** cannot obtain enough memory to keep its tables. If you do not specify **-t** and **fsck** needs a scratch file, it prompts you for the name of the scratch file. However, if you have specified the **-p** flag, **fsck** fails. The file chosen must not be on the file system being checked. If it is not a special file, it is removed when **fsck** ends.
-y	Assumes a `yes` response to all questions asked by **fsck**. This lets **fsck** take any action that it considers necessary. Use this flag only on severely damaged file systems.

Examples

1. To check all the default file systems:

   ```
   fsck
   ```

   This checks all the file systems marked `check=true` in **/etc/filesystems**. This form of the **fsck** command asks you for permission before making any changes to a file system.

2. To fix minor problems with the default file systems automatically:

   ```
   fsck -p
   ```

3. To check a specific file system:

   ```
   fsck /dev/hd1
   ```

   This checks the unmounted file system located on the `/dev/hd1` device.

4. To simultaneously check two file systems on two different drives:

   ```
   dfsck  -p /dev/hd1  -  -p /dev/hd7
   ```

   This checks both file systems simultaneously, if the file systems on the devices `/dev/hd1` and `/dev/hd7` are located on two different drives. You can also specify the file system names that are found in the **/etc/filesystems** file.

Files

/etc/filesystems	Contains default list of file systems to check.

Related Information

The following commands: "**rc**" on page 806, "**fsdb**" on page 450, "**istat**" on page 545, "**mkfs**" on page 658, "**ncheck**" on page 683, and "**shutdown**" on page 946.

The **filesystems** and **fs** files in *AIX Operating System Technical Reference.*

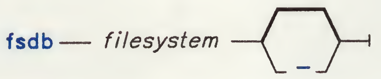
The discussion of **fsck** and **dfsck** in *Managing the AIX Operating System*

fsdb

Purpose

Debugs file systems.

Syntax



OL805244

Description

Warning: This program is not intended for use with diskette-based file systems because of the difference in superblock structure and the general format of the file system.

You can use the **fsdb** command to examine and patch a damaged file system after a system crash. It allows you to access blocks and i-numbers and to examine various parts of an i-node. You can reference components of the i-node symbolically. These features simplify procedures for correcting control-block entries or for descending the file-system tree.

The file system to be examined can be specified by a block device name, a raw device name, or a mounted file system name. In the latter case, **fsdb** determines the associated file name by reading the file **/etc/filesystems.**

Any numbers you enter are considered decimal by default, unless you prefix them with a 0 (zero) to indicate an octal number.

Because **fsdb** reads and writes one block at a time, it works with raw as well as with block I/O. It uses a buffer management routine to retain commonly used blocks of data in order to reduce the number of **read** system calls. All assignment operations write the corresponding block immediately.

Flag

-	Disables the error checking routines used to verify i-node and block addresses. The **O** subcommand toggles these routines on and off. When these routines are running, **fsdb** reads the i-size and f-size entries from the superblock of the file system.

Subcommands

The subcommands you give to **fsdb** are requests to display or modify information. A display subcommand is a block address optionally followed by a display format specification. A field modification subcommand is similar to the display subcommand but may include a subfield specification, an operator, and a value. An address specification is a number optionally followed by a type specifier and subfield specification.

The display subcommands are:

num	Display data at absolute address *num*.
*i-number***i**	Display data at *i-number*.
*block-address***b**	Display data at *block-address*.
*directory-slot-offset***d**	Display data at *directory-slot-offset*.
q	Quit.
!	Escape to the shell.

The display formats are:

p	General display facilities
f	File display facility.

You can step through the i-node information examining each byte, word, or double word. Select the desired display mode by entering one of the following subcommands:

B	Begin displaying in byte mode.
D	Begin displaying in double-word mode.
W	Begin displaying in word mode.
O	Toggle error checking on or off.

Moving forward or backward through the i-node data is done with the following symbols:

+*num*	Move forward the specified number of units currently in effect.
-*num*	Move backward the specified number of units currently in effect.

The following symbols allow you to store the current address and return to it conveniently:

>*address*	Store *address* for later reference. If you do not specify *address*, **fsdb** stores the current address.
<	Return to the previously stored address.

The display format applied to the information at the selected address is the one currently in effect. You may receive an error message indicating improper alignment if the address you specify does not fall on an even boundary.

The display facilities display a formatted output in various styles. The current address is normalized to an appropriate boundary before display begins. The boundary advances with display and is left at the address of the last item displayed. The output can be ended at any time by pressing INTERRUPT (**Alt-Pause**).

If you enter a number after the **p** symbol, **fsdb** displays that number of entries. A check is made to detect block boundary overflows because logically sequential blocks are generally not physically sequential. If you enter a count of zero, **fsdb** displays all entries to the end of the current block.

The display formats available are:

i	Display as i-nodes.
d	Display as directories.
o	Display as octal words.
e	Display as decimal words.
c	Display as characters.
b	Display as octal bytes.
y	Display as hex bytes.

Use the **f** symbol to display data blocks associated with the current i-node. If you enter a number after **f**, **fsdb** displays that block of the file. Block numbering begins at zero. The desired display subcommand follows the block number, if present, or the **f** symbol. The display facility works for large as well as small files. It checks for special devices and also checks the data are not zero.

You can use dots (.), tabs, and spaces as subcommand delimiters, but they are not necessary. Pressing just the **Enter** key (entering a blank line) increments the current address by the size of the data type last displayed. That is, the address is set to the next byte, word, double word, directory entry or i-node, allowing you to step through a region of a file system. **fsdb** displays information in a format appropriate to the data type. Bytes, words and double words are displayed as an octal address followed by the octal representation of the data at that address and the decimal equivalent enclosed in parentheses. **fsdb** adds a **.B** or **.D** to the end of the address to indicate a display of byte or double word values. It displays directories as a directory slot offset followed by the decimal i-number and the character representation of the entry name. It displays i-nodes with labeled fields describing each element. The environment variables **NLLDATE** and **NLTIME** control the formats of the date and time.

The following mnemonics are used for the names of the fields of an i-node and refer to the current working i-node:

md	Permission mode
ln	Link count
uid	User number
gid	Group number
sz	File size
a*n*	Data block numbers (0 - 12)
at	Access time
mt	Modification time
maj	Major device number
min	Minor device number.

The general form for assigning new values is:

mnemonic operator new-value

The **fsdb** command modifies the value of the field specified by *mnemonic* according to the *operator* and *new-value*.

Valid operators include:

=	Assign *new-value* to the specified *mnemonic*.
=+	Increment the *mnemonic* by the specified *new-value*. The default *new-value* is 1.
=-	Decrease the *mnemonic* by the specified *new-value*. The default *new-value* is 1.
="	Assign character string *new-value* to the specified *mnemonic*.

Examples

The following examples show subcommands that you can use after starting **fsdb**.

1. To display an i-node:

   ```
   386i
   ```

   This displays i-number 386 in i-node format. It now becomes the current i-node.

2. To change the link count for the current i-node to 4:

   ```
   ln=4
   ```

3. To increase the link count of the current i-node by 1:

   ```
   ln=+1
   ```

4. To display part of the file associated with the current i-node:

   ```
   fc
   ```

   This displays as ASCII text block zero of the file associated with the current i-node.

5. To display entries of a directory:

   ```
   2i.fd
   ```

   This changes the current i-node to the root i-node (i-node 2), then displays the directory entries in the first block associated with that i-node.

6. To go down a level of the directory tree:

   ```
   d5i.fc
   ```

   This changes the current i-node to the one associated with directory entry 5. Then it displays the first block of the file as ASCII text (fc). Directory entries are numbered starting from 0 (zero).

7. To display a block when you know its block number:

   ```
   1b.p0o
   ```

   This displays the superblock (block 1) of file system in octal.

8. To change the i-number of a directory entry:

   ```
   2i.a0b.d7=3
   ```

   This changes the i-number of directory entry 7 in the root directory (2i) to 3. This example also shows how several operations can be combined on one line.

9. To change the file name of a directory entry:

   ```
   d7.nm="chap1.rec"
   ```

   This changes the name field of directory entry 7 to chap1.rec.

10. To display a given block of the file associated with the current i-node:

    ```
    a2b.p0d
    ```

    This displays block 2 of the current i-node as directory entries.

Related Information

The following command: "**fsck**, **dfsck**" on page 445.

The **fs** and **dir** files and the **environment** miscellaneous facility in *AIX Operating System Technical Reference.*

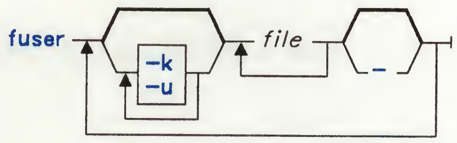
"Overview of International Character Support" in *Managing the AIX Operating System.*

fuser

Purpose

Identifies processes using a file or file structure.

Syntax



OL805055

Description

The **fuser** command lists, for local processes, the process numbers of the processes using the specified local or remote *file*. For remote processes that use local *files*, **fuser** lists the node (NID) that has the files open. It does not list the process numbers, user names, or usage information. For block special devices, all processes using any file on that device are listed. The process number is followed by a letter indicating how the process is using the file:

c	Using *file* as the current directory
p	Using *file* as the parent of the current directory (only when in use by the system)
r	Using *file* as the root directory.

The process numbers are written as a single line to standard output, separated by spaces and ended with a single new-line character. All other output is written to standard error.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-k Sends the **SIGKILL** signal to each local process. Only the person operating with superuser authority can kill another user's process (see "**kill**" on page 552). **SIGKILL** is not sent to remote processes.

-u Indicates the login name in parentheses after the process number. The login name is not listed for remote processes.

- Cancels any flags selected for the previous set of file or files.

Flags may be respecified between groups of files on the command line. The new set of flags replaces the old set.

Examples

1. To list the ID numbers of the processes using the **/etc/passwd** file:

   ```
   fuser  /etc/passwd
   ```

2. To list the process IDs and user names of the processes using the **/etc/filesystems** file:

   ```
   fuser  -u  /etc/filesystems
   ```

3. To stop all of the processes using a given disk drive:

   ```
   fuser  -k  -u  /dev/hd1
   ```

   This lists the process ID and user name, and then stops each process that is using the /dev/hd1 disk drive. You must have superuser authority to stop processes that belong to someone else. You might want to do this if you are trying to unmount /dev/hd1, and a process accessing it is preventing you from doing so.

4. To perform the actions of the previous examples in reverse order:

   ```
   fuser  -k  -u  /dev/hd1  -  -u  /etc/filesystems  -  /etc/passwd
   ```

   Note that lone dashes before the -u and before /etc/passwd turn off both the -k and -u flags.

Files

/unix	System kernel image.
/dev/kmem	For system image.
/dev/mem	Also for system image.

Related Information

The following commands: "**killall**" on page 555, "**mount**" on page 669, and "**ps**" on page 786.

The **kill** and **signal** system calls in *AIX Operating System Technical Reference.*

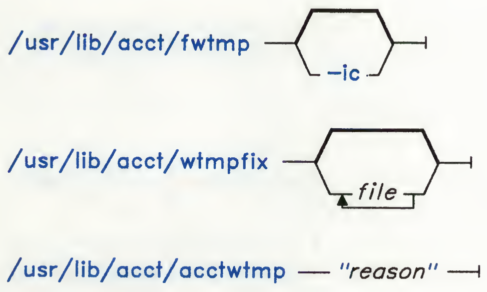
"Using Distributed Services" in *Managing the AIX Operating System.*

fwtmp

Purpose

Manipulates connect accounting records.

Syntax



OL805239

Description

fwtmp

The **fwtmp** command reads **wtmp** records from standard input and converts them to formatted ASCII records, which it writes to standard output.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flag

-ic Reads ASCII input and writes output in binary form.

acctwtmp

The **acctwtmp** command writes to standard output a **utmp** record containing the string *reason* and the current date and time. A *reason* can contain 11 or fewer characters.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

wtmpfix

The **wtmpfix** command examines standard input or the named *file*s containing records in **wtmp** format, corrects the date and time stamps to make the entries consistent, and writes the corrected input to standard output. (It is necessary that date and time stamps be consistent because **acctcon1** generates an error and stops when it encounters inconsistent date change records.)

Each time the date is set (on system startup or with the **date** command) a pair of date change records is written to **/usr/adm/wtmp**. The first record is the old date, denoted by the string **old time** placed in the line field and the flag **OLD_TIME** placed in the type field. The second record is the new date, denoted by the string **new time** placed in the line field and the flag **NEW_TIME** placed in the type field. The **wtmpfix** command uses these records to synchronize all date and time stamps in the file.

In addition to correcting date and time stamps, **wtmpfix** checks the validity of the name field to ensure that it consists solely of alphanumeric characters, a dollar sign ($), or spaces. If it encounters an invalid name, it changes the login name to **INVALID** and writes a diagnostic to standard error. In this way, **wtmpfix** reduces the chance that **acctcon2** will fail when it processes connect accounting records.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Files

/usr/adm/wtmp	Contains records of date changes that include an old date and a new date.
/usr/include/utmp.h	Contains history records that include a reason, date, and time.

Related Information

The following commands: “**acct/***” on page 13, “**acctcms**” on page 18, “**acctcom**” on page 20, “**acctcon**” on page 24, “**acctdisk, acctdusg**” on page 26, “**acctmerg**” on page 28, “**acctprc**” on page 30, and “**runacct**” on page 848.

The **acct** system call and the **acct** and **utmp** files in *AIX Operating System Technical Reference.*

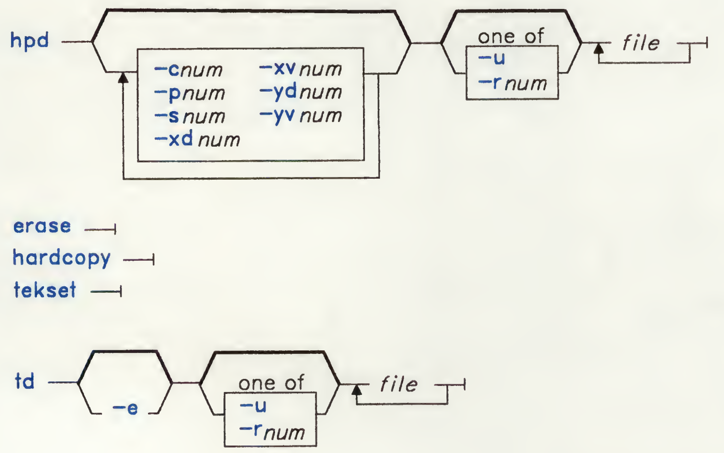
“Running System Accounting” in *Managing the AIX Operating System.*

gdev

Purpose

Provides graphical device routines and filters.

Syntax



OL777036

Description

The following commands provide various graphical device routines and filters. They all reside in the **/usr/bin/graf** directory (see "**graphics**" on page 497).

hpd

The **hpd** command takes a graphical file in **gps** format (standard input by default), and translates it into instructions for the Hewlett-Packard 7221A Graphics Plotter. See the **gps** file in *AIX Operating System Technical Reference* for a description of this file format. It computes a viewing window from the maximum and minimum points in the file, unless you specify the **-r** or **-u** flags.